分散メモリ型並列計算機における
タスクスケジューリングに関する研究

1999年12月

藤本 典幸

内容梗概

並列処理における重要な基本問題の1つにタスクスケジューリング問題(以降TSP)がある．TSPは，並列計算を表す重み付きのDAG(Directed Acyclic Graph) Gと利用可能なプロセッサ数pが与えられたときに，p台のプロセッサを用いてGが表す並列計算を計算するためには，すべてのタスクについて，そのタスクをいつ，どのプロセッサで実行すればよいかを表すスケジュールを求める問題である．ここでGの節点はタスクを表し，有向辺はタスク間の実行順序の依存関係を表す．節点の重みと有向辺の重みはそれぞれタスクの処理時間とタスクの計算結果がそれを必要とするタスクで利用可能になるまでの通信遅延時間を表す．Gの有向辺の重みがすべて定数τである場合のTSPをTSP_τと呼ぶ．スケジュールですべてのタスクの計算が完了するまでの時間をそのスケジュールのメイクスパンという．TSP_τでメイクスパン最小のスケジュールを求める問題はNP困難である．とくに，$\tau = 0$かつ$p = 2$に限定した場合や，pを任意に設定でき，かつ，すべての節点の重みが等しいと限定した場合でもNP困難である．このためTSP_τに対して様々な近似アルゴリズムが考えられている．本研究では，TSP_τに関する2つの問題を扱う．

本研究では，まず，バルク同期スケジュールを提案し，入力としてG, p, τがそれぞれ任意に与えられたときに，メイクスパンの小さいバルク同期スケジュールを出力する問題に対するアルゴリズムBCSHを示す．バルク同期スケジュールとは，通信を行わず計算のみを行う計算フェーズと計算を行わず通信のみを行う通信フェーズのみが交互に現れるという特徴をもつスケジュールである．バルク同期スケジュールには現在主流の並列計算機アーキテクチャである分散メモリ型並列計算機上で並列プログラム化したときに，通信を効率よく実装できるという利点がある．すなわちメイクスパンの小さいバルク同期スケジュールを生成できれば，分散メモリ型並列計算機用の高速な並列プログラムを生成できる．既存の研究でアルゴリズムの評価に用いるDAGは，プログラムとして意味のないランダムグラフであるかまたは実際のプログラムの計算を表すDAGを用いている場合でも，その節点数が数百程度（問題サイズ数十に相当）であり，分散メモリ型並列計算機上での並列計算の対象としては小さかった．

これに対して本研究では，プログラムからDAGを機械的に生成することにより，実験には節点数数十万規模（問題サイズ数百から数千に相当）の実際のプログラムの計算を表すDAGを用いた．LU分解，高速フーリエ変換，連立一次方程式の反復解法のヤコビ法について，分散メモリ型並列計算機である富士通AP1000上で，BCSHが生成するスケジュール通りに動作する並列プログラムを生成し，プロセッサをp台用いたときにプロセッサを1台用いる場合にくらべて並列プログラムの速度がどれくらい向上するか（速度向上率）を評価した．速度向上率の理想値はpである．評価実験の結果，BCSHを用いれば，32台のプロセッサを用いたとき，問題サイズ128のLU分解で速度向上率8倍，問題サイズ8192の高速フーリエ変換で速度向上率12倍，問題サイズ512のヤコビ法で速度向上率18倍の性能をもつ並列プログラムを生成できることがわかった．

次に本研究では，各タスクの処理時間がすべて等しく，DAGが完全2分木である場合のTSP_τに対するアルゴリズムを示す．完全2分木は並列プログラム中でサブルーチンとしてよく用いられるリダクション演算を表せる．すなわち完全2分木に対してメイクスパンの小さいスケジュールが得られれば，リダクション演算を高速に実行するライブラリを実現できる．本研究では，まず，プロセッサ数が任意に設定できる場合にこの問題に対して2つのアルゴリズムALG1，ALG2を提案し，2から10000の範囲のτと，ノード数100万以内の完全2分木に対して，ALG1とALG2が生成するスケジュールのメイクスパンを実験により評価する．アルゴリズムが生成するスケジュールのメイクスパンの最適スケジュールのメイクスパンに対する比率をそのアルゴリズムの近似精度という．ALG1の近似精度は平均1.1から1.3以内であること，プロセッサ割当が非常に単純であるにもかかわらず，ALG2の近似精度は平均1.1から1.4以内であることを示す．次に利用できるプロセッサ数が限られる場合のアルゴリズムALG3を提案し，ALG3の近似精度が高々$\frac{10}{3}$であることを解析的に示す．さらに，任意のDAGとτに対してスケジュールのメイクスパンの既知の下界の改良方法を提案し，完全2分木，バタフライ，ダイアモンドについて既知の下界が改良できることを示す．

関連発表論文

(1) 論文

[1-1] 藤本典幸，柘植宗俊，萩原兼一，首藤勝："プロセッサ間通信遅延を考慮した完全２分木状タスク依存グラフのスケジューリングアルゴリズム"，電子情報通信学会論文誌，Vol.J80-D-I, No.4, pp.369-379 (1997)

[1-2] 柘植宗俊，藤本典幸，萩原兼一："平衡２分探索木に対する並列オンライン操作"，電子情報通信学会論文誌，Vol.J80-D-I, No.7, pp.582-590 (1997)

[1-3] 伊野 文彦，杉野 陽一，山崎 良太，藤本 典幸，萩原 兼一："並列プログラムの性能改善支援機能をもつ性能解析システム：Gordini"，情報処理学会論文誌，条件付き採録 (1999年 11 月)

[1-4] 藤本典幸，馬場大樹，橋本貴至，萩原兼一："BSP モデルに基づくスケジュールを生成するタスクスケジューリングアルゴリズム"，情報処理学会論文誌 (投稿中)，(1999)

(2) 国際会議等

[2-1] Noriyuki Fujimoto, Tomoki Baba, Takashi Hashimoto, Kenichi Hagihara: "A Task Scheduling Algorithm to Package Messages on Distributed Memory Parallel Machines", in Proc. of 1999 International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN'99), pp. 236-241 , Fremantle, Australia, June (1999)

[2-2] Noriyuki Fujimoto, Takashi Hashimoto, Masahiro Mori, Kenichi Hagihara: "On the Performance Gap between a Task Schedule and Its Corresponding Parallel Program", in Proc. of 1999 International Workshop on Parallel and Distributed Computing for Symbolic and Irregular Applications, to be published, Sendai, Japan, July (1999)

(3) 研究会報告等

[3-1] 藤本典幸，柘植宗俊，萩原兼一，魚井宏高，首藤勝："プロセッサ間通信遅延を考慮した完全２分木状タスク依存グラフのスケジューリング手法"，電子情報通信学会技術研究報告，COMP 93-95,pp.65-72(1994)

[3-2] 萩原兼一，藤本典幸，柘植宗俊："プロセッサ間通信遅延を考慮したタスクスケジューリング（完全２分木の場合）"，文部省重点領域研究「超並列処理に基づく情報処理基本体系」第４回シンポジウム予稿集,pp.(2-230)-(2-244)(1994)

[3-3] 日隈康, 橋本貴至, 藤本典幸, 萩原兼一："通信の一括化に適したタスクスケジューリングアルゴリズム", 電子情報通信学会技術研究報告，COMP97-78, pp.1-8(1998)

[3-4] 藤本典幸, 日隈 康, 橋本 貴至, 馬場 大樹, 萩原 兼一："分散メモリ型マシンの通信特性を考慮したタスクスケジューリングアルゴリズムのコンパイラへの応用について", PCW'98 Japan 予稿集, P-A, pp.1-8 (1998)

[3-5] 馬場大樹, 橋本貴至, 藤本典幸, 萩原兼一："分散メモリ型並列計算機の通信特性を考慮したタスクスケジューリングアルゴリズムの開発とその評価", 電子情報通信学会技術研究報告，COMP98-72, pp.1-8 (1999)

[3-6] 森雅博, 橋本貴至, 西村晃一, 藤本典幸, 萩原兼一："タスク複製率とプロセッサアイドル率に着目した BSP スケジュール生成手法の提案", 情報処理学会研究会報告，2000-AL-71，pp.9-16 (2000)

[3-7] 橋本貴至，森雅博，西村晃一，藤本典幸，萩原兼一："タスクスケジューリングを用いた並列プログラム生成におけるタスク粒度の調整とその評価", 情報処理学会研究会報告，2000-AL-71，pp.17-24 (2000)

目 次

第1章　緒論

並列処理における重要な基本問題の1つにタスクスケジューリング問題（以降TSP）がある．TSPは，並列計算を表す重み付きのDAG(Directed Acyclic Graph) G と利用可能なプロセッサ数 p が与えられたときに，p 台のプロセッサを用いて G が表す並列計算を計算するためには，すべてのタスクについて，そのタスクをいつ，どのプロセッサで実行すればよいかを表すスケジュールを求める問題である．ここで G の節点はタスクを表し，有向辺はタスク間の実行順序の依存関係を表す．節点の重みと有向辺の重みはそれぞれタスクの処理時間とタスクの計算結果がそれを必要とするタスクで利用可能になるまでの通信遅延時間を表す．スケジュールですべてのタスクの計算が完了するまでの時間をそのスケジュールのメイクスパンという．メイクスパン最小のスケジュールを求める問題を最適TSPと呼ぶ．最適スケジュールを多項式時間で得られるのは表1に示したごく制限された場合についてのみであり，DAGのトポロジなどに制限のない一般の場合については，最適TSPはNP困難である[4].

並列計算を実行する並列計算機において，どのようなパターンのプロセッサ間通信を行なっても，高々一律 τ 単位時間の後には，送信されたデータが受信プロセッサのタスクで利用可能になるという形で通信遅延を考慮したTSP[17]を TSP_τ と呼ぶ．TSP_τ でメイクスパン最小のスケジュールを求める問題はNP困難である．とくに，$\tau = 0$ かつ $p = 2$ に限定した場合や，p を任意に設定でき，かつ，すべてのタスクの処理時間が単位時間であると限定した場合[17]でもNP困難である．このため TSP_τ

表1.1: 最適スケジュールが多項式時間で得られる場合（v はDAGの節点数，e はDAGの有向辺の数，α は節点の多項式関数[6]）

通信遅延	DAG	タスクの計算時間	プロセッサ数	計算量
すべて0	木	すべて等しい	任意	$O(v)$
すべて0	期間順序[4]	すべて等しい	任意	$O(v)$
すべて0	任意	すべて等しい	2	$O(e + v\alpha(v))$

に対して様々な近似アルゴリズムが考えられている [13, 17, 18]. 本研究では, TSP_τ に関する2つの問題を扱う.

メッセージベクトル化などの通信の一括化を用いれば, 分散メモリ型並列計算機 (以降, 分散メモリマシン) 上での並列プログラムの性能を著しく改善できることはよく知られている [3, 21]. しかし並列プログラム化したときに通信の一括化が行いやすい性質をもつスケジュールを生成するタスクスケジューリングアルゴリズムはこれまで存在しなかった. 分散メモリマシンを対象にした TSP に関する既存の研究は, 単に通信遅延時間が1タスクの実行時間の数百から数百万倍の値になるという形でしか分散メモリマシンの特性を考慮していなかった [1, 15, 17]. したがって既存のアルゴリズムが生成したスケジュール通りに動作する並列プログラムを作成しても, 分散メモリマシン上でのその性能は必ずしもよくない. 例えば図 1.1 は, 性能のよいスケジュールを生成するとされている既存のアルゴリズム DSH[11] が生成したスケジュールの速度向上率と, DSH が生成したスケジュール通りに動作する並列プログラム分散メモリマシン上での速度向上率を示したものである. ここで速度向上率とは, プロセッサを p 台用いたときに, プロセッサを1台だけ用いる場合にくらべてプログラムあるいはスケジュールの実行速度がどれくらい向上するかを表す比率である. DSH の場合, スケジュールの速度向上率とプログラムの速度向上率の間には大きな隔たりがある. 一方, Valiant が提案した並列計算モデルである BSP モデル [20] に基づく並列プログラムには, 通信の一括化が適用しやすい [19]. そこで本研究では, まず, バルク同期スケジュールを提案し, 入力として G, p, τ が任意に与えられたときに, メイクスパンの小さいバルク同期スケジュールを出力する問題に対するアルゴリズムを示す. バルク同期スケジュールとは, BSP モデルに基づく並列計算を通信の一括化が適用しやすいという利点を失うことなく簡単化したスケジュールで, 通信を行わず計算のみを行う計算フェーズと計算を行わず通信のみを行う通信フェーズのみが交互に現れるという特徴をもつ. メイクスパンの小さいバルク同期スケジュールを生成できれば, 分散メモリ型並列計算機用の実行時間が短い並列プログラムを生成できる. この目的のために, 本研究では, まずバルク同期スケジュールを定義する. BSP モデルの定義は Valiant により与えられているが, バルク同期スケジュールという概念はこれまでなかった. BSP モデルに基づいた並列計算も計算フェーズと通信フェーズのみが交互に現れるという特徴をもつが, BSP モデルでは通信フェーズ部分のメイクスパンは, 有向辺の重みだけでなく, 通信パターンにも依存する. このため BSP モデル上でタスクスケジューリング問題を解くよりも提案する手法

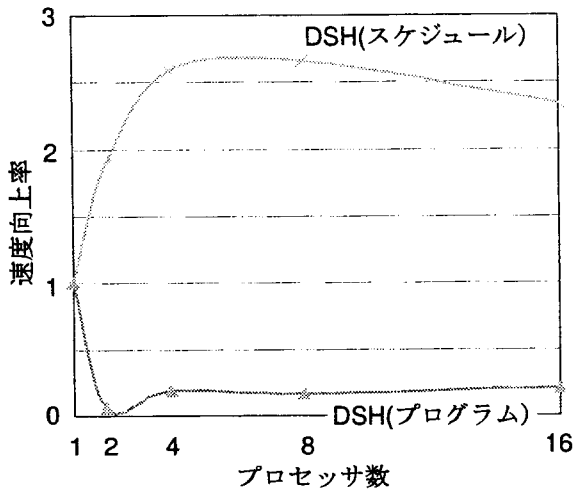


図 1.1: 既存アルゴリズム DSH のスケジュール上の性能と分散メモリマシン上での性能 (問題サイズ 32 の LU 分解，AP1000)

を用いる方が，より簡単に通信の一括化が適用しやすいスケジュールを得ることができる．次に，並列プログラム化したときに実行時間が小さいバルク同期スケジュールのもつ性質を考察し，その性質を満たすバルク同期スケジュールを生成するアルゴリズム BCSH (Bulk Communication Scheduling Heuristic) を提案する．

既存の研究でアルゴリズムの評価に用いる DAG は，プログラムとして意味のないランダムグラフであるか [1, 15]，または実際のプログラムの計算を表す DAG を用いている場合でも，その節点数が数百程度（問題サイズ数十に相当）であり [1, 8, 10]，分散メモリマシン上での並列計算の対象としては小さかった．これに対して本研究では，プログラムから DAG を機械的に生成することにより，実験には節点数数十万規模（問題サイズ数百から数千に相当）の実際のプログラムの計算を表す DAG を用いた．LU 分解，高速フーリエ変換，連立一次方程式の反復解法のヤコビ法について，分散メモリ型並列計算機である富士通 AP1000 上で，BCSH が生成するスケジュール通りに動作する並列プログラムを生成し，プロセッサを p 台用いたときにプロセッサを 1 台用いる場合にくらべて並列プログラムの速度がどれくらい向上するか（速度向上率）を評価した．速度向上率の理想値は p である．評価実験の結果，BCSH を用いれば，32 台のプロセッサを用いたとき，問題サイズ 128 の LU 分解で速度向上率 8 倍，問題サイズ 8192 の高速フーリエ変換で速度向上率 12 倍，問題サ

イズ512のヤコビ法で速度向上率18倍の性能をもつ並列プログラムを生成できることがわかった.

次に本研究では,DAGがタスクの処理時間がすべて等しい完全2分木である場合のTSP_τに対するアルゴリズムを示す.N個のデータの総和を求める演算,最小値を求める演算など,複数のデータに同一の2項演算を施して単一の結果を得る演算をリダクション演算という.リダクション演算は並列プログラム中でサブルーチンとしてよく用いられる.タスクの処理時間がすべて等しい完全2分木はリダクション演算を表せる.すなわち完全2分木に対してメイクスパンの小さいスケジュールが得られれば,リダクション演算を高速に実行するライブラリを実現できる.

本研究では,まず,プロセッサ数が任意に設定できる場合にこの問題に対して2つのアルゴリズムALG1,ALG2を提案する.次に本研究では,提案するアルゴリズムのメイクスパンの最適スケジュールのメイクスパンに対する比率をより正確に評価するために,任意のDAGとτに対してスケジュールのメイクスパンの既知の下界の改良方法を提案し,完全2分木,バタフライ,ダイアモンドについて既知の下界が改良できることを示す.スケジュールのメイクスパンの下界がLである場合にメイクスパンMのスケジュールSが得られたとすると,Sのメイクスパンは最適スケジュールのメイクスパンの高々M/L倍である.この改良した下界を用いて,2から10000の範囲のτと,ノード数100万以内の完全2分木に対して,ALG1とALG2が生成するスケジュールのメイクスパンを実験により評価する.アルゴリズムが生成するスケジュールのメイクスパンの最適スケジュールのメイクスパンに対する比率をそのアルゴリズムの近似精度という.ALG1の近似精度は平均1.1から1.3以内であること,プロセッサ割当が非常に単純であるにもかかわらず,ALG2の近似精度は平均1.1から1.4以内であることを示す.次に利用できるプロセッサ数が限られる場合のアルゴリズムALG3を提案し,ALG3の近似精度が高々$\frac{10}{3}$であることを解析的に示す.

本論文の構成は以下の通りである.まず2章ではタスクスケジューリング問題や通信の一括化などを定義する.そして3章でバルク同期スケジュールを生成するアルゴリズムBCSHを提案し,BCSHが生成するスケジュール通りに動作する並列プログラムの分散メモリマシン上での性能を評価する.次に4章で各タスクの処理時間が等しい場合の完全2分木のタスクスケジューリングアルゴリズムALG1,ALG2,ALG3を提案し,スケジュールのメイクスパンの既知の下界の改良により提案するアルゴリズムが生成するスケジュールと最適スケジュールのメイクスパンを比較評価する.最後に5章でまとめと今後の課題を示す.

第2章　諸定義

2.1　タスクグラフ

重み付きのDAG(Directed Acyclic Graph)を$G=(V,E,\lambda,\tau)$と書く．ここでVは節点の集合，Eは有向辺の集合，λは節点から節点の重みへの関数，τは有向辺から有向辺の重みへの関数である．節点uから節点vへの有向辺を(u,v)と書く．並列計算をモデル化したDAGをタスクグラフと呼ぶ．図2.1にタスクグラフの例を示す．タスクグラフの節点は並列計算のタスクを表す．節点uが表すタスクをT_uと書く．値$\lambda(u)$は，T_uの実行時間が$\lambda(u)$単位時間であることを意味する．有向辺(u,v)は，T_vの計算のためにはT_uの計算結果が必要であることを意味する．値$\tau(u,v)$は，T_uを計算するプロセッサpからT_vを計算するプロセッサqへのプロセッサ間通信遅延は，pとqが異なる場合，高々$\tau(u,v)$単位時間であることを意味する．pとqが同一なら，通信遅延は0とする．図2.1にタスクグラフの例を示す．タスクT_3の実行時間は2単位時間であり，T_5を計算するためには，T_3とT_4の計算結果が必要である．またT_5とT_3を異なるプロセッサで計算する場合，T_3の計算結果をT_5を計算するプロセッサに送信するために必要な通信遅延時間は2単位時間である．Gのすべてのタスクの処理時間が単位時間のとき，Gは**均質**であるという．$(u,v)\in E$のとき，uはvの**直接先行節点**，vはuの**直接後続節点**という．$(v_i,v_{i+1})\in E(1\leq i\leq k)$のとき，節点の系列$\langle v_1,v_2,\cdots,v_k\rangle$を$G$

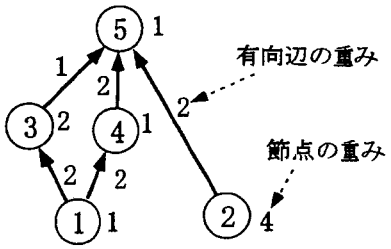


図 2.1: タスクグラフの例

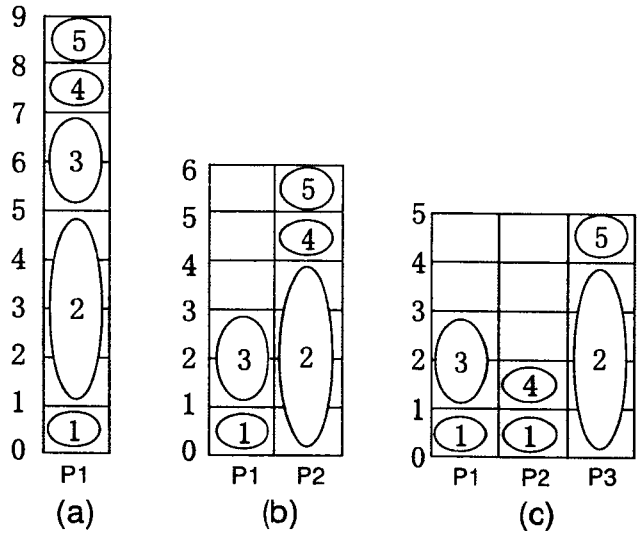


図 2.2: スケジュールの例

の**有向道**という．ここで k を**有向道の長さ**という．G において，節点 u から節点 v にいたる有向道があるとき，u は v の**子孫**であるという．とくにその有向道の長さが 1 のとき u は v の**子**という．v 自身は v の子ではない．本論文では，すべての有向辺が葉から根の方向をもつ有向完全２分木を単に**完全２分木**と呼ぶ．また**完全２分木の高さ**を葉から根にいたる有向道上にある節点の数と定義する．例えば，図 2.3 の 8 と 9 は 4 の子であり，かつ 2 の子孫で，完全２分木の高さは 4 である．高さ h の完全２分木を C_h と書く．

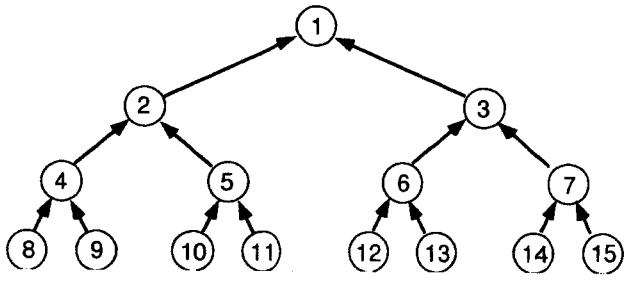


図 2.3: 完全２分木の例

2.2 スケジュール

Thurimellaらは各タスクの実行時間が単位時間であり，かつ，すべてのタスク間通信遅延時間が等しい場合についてスケジュールの定義を与えた[18]．本論文ではThurimellaらの定義を拡張し，各タスクの実行時間と各タスク間の通信遅延時間が任意である一般の場合についてのスケジュールの定義を与える．

利用可能なプロセッサ数がpの場合の$G=(V,E,\tau,\lambda)$の**スケジュール**とは，以下の条件R1，R2，R3を満たす3つ組$\langle v,q,t\rangle$の有限集合をいう．ここで，$v\in V$，qは$1\leq q\leq p$を満たす整数，tは非負整数である．3つ組$\langle v,q,t\rangle\in S$は，プロセッサ$q$がタスク$T_v$を時刻$t$から時刻$t+\lambda(v)$の間に計算することを意味する．

R1 各$v\in V$に対して，3つ組$\langle v,q,t\rangle$が少なくとも1つSに存在する．(各タスクT_vは少なくとも1回実行される)

R2 $t\leq t'\leq t+\lambda(v)$となる2つの3つ組$\langle v,q,t\rangle,\langle v',q,t'\rangle$は$S$に存在しない．(プロセッサは任意の時刻に高々1つのタスクしか実行できない)

R3 $(u,v)\in E$かつ$\langle v,q,t\rangle\in S$ならば$t'\leq t-\lambda(u)$となる3つ組$\langle u,q,t'\rangle$が$S$に存在するか，または$t''\leq t-\lambda(u)-\tau(u,v)$かつ$q\neq q'$となる3つ組$\langle u,q',t''\rangle$が$S$に存在する．($T_v$が$T_u$の結果に依存する場合，$T_u$と$T_v$は同じプロセッサに割り当てられて$T_u$は$T_v$がスケジュールされる時刻より前に実行完了されるか，あるいはT_uはT_vがスケジュールされる時刻よりも少なくとも$\tau(u,v)$単位時間前に異なるプロセッサで実行完了される)

図2.1のタスクグラフを1台，2台，3台のプロセッサにスケジュールした例をそれぞれ図2.2(a)，(b)，(c)に示す．図2.2(b)は，スケジュール$\{(1,1,0),(2,2,0),(3,1,1),(4,2,4),(5,2,5)\}$を縦軸に時刻，横軸にプロセッサ番号をとった図で表している．スケジュールを表す図2.2 (a)，(b)，(c)のような図を**ガント図**[4]という．以降，本論文ではスケジュールをガント図を用いて表す．スケジュールSのすべてのタスクの実行が完了する時刻$\max\{t+\lambda(v)|\langle v,q,t\rangle\in S\}$を$S$の**メイクスパン**という．例えば図2.2(b)のスケジュールのメイクスパンは6である．スケジュールSの3つ組に現れる異なるプロセッサの数をS**の必要プロセッサ数**という．

2.3 タスクスケジューリング問題

タスクスケジューリング問題は，タスクグラフGと利用可能なプロセッサ数pが与えられるとき，Gとpに対するスケジュールを1つ見つける問題である．Gのすべての有向辺の重みが一律τ単位時間である場合のタスクスケジューリング問題をTSP_τと呼ぶ．**最適タスクスケジューリング問題**は，タスクグラフGと利用可能なプロセッサ数pが与えられるとき，Gとpに対するすべてのスケジュールの中で最小のメイクスパンを持つスケジュールを1つ見つける問題である．このメイクスパン最小のスケジュールをタスクスケジューリング問題の**最適解**という．pの値が自由に設定できる場合に，均質なGと通信遅延τが与えられて，Gのスケジュールを1つ求める問題を$STSP_\tau$という．

2.4 タスク複製

あるタスクvに対して$\langle v,p,s\rangle$，$\langle v,q,t\rangle \in S(p \neq q)$となるような3つ組が存在するとき，$v$は$S$において**再計算**されるという[18]．再計算によりタスクを重複して複数のプロセッサで実行することを**タスク複製**という[11]．また重複して実行されるタスクを**複製タスク**と呼ぶ．プロセッサでタスクを計算する際に通信が必要になるのは，そのタスクの計算に必要な他のタスクの計算結果をそのプロセッサで計算していない場合である．2つの直接後続節点v_1, v_2をもつ節点uを考える．タスク複製を許さない場合，v_1またはv_2のいずれかの計算には必ず通信が必要になる．しかしuを複製してv_1，v_2を計算するプロセッサの両方で実行すればv_1，v_2の計算の際には通信が不要となる．このように，タスク複製には通信を減らす効果がある[11]．

図2.4にタスク複製の例を示す．T_5をP_1とP_2で重複して実行することにより，P_2上でのT_3の実行開始時刻が早められている．このためT_4とT_3の間の並列性が完全に利用できている．また，T_4とT_3を同じプロセッサに割り当てる場合にくらべてT_1の実行時間を早められている．T_3は$P1$で実行するために複製されているのでT_2のための通信遅延は必要ない．

2.5 独立グループ集合

タスクグラフ$G=(V,E,\lambda,\tau)$の任意の連結成分を$G'=(V',E',\lambda,\tau)$とする．V'をGの**節点グループ**と呼ぶ．C_jとC_kをそれぞれGの節点

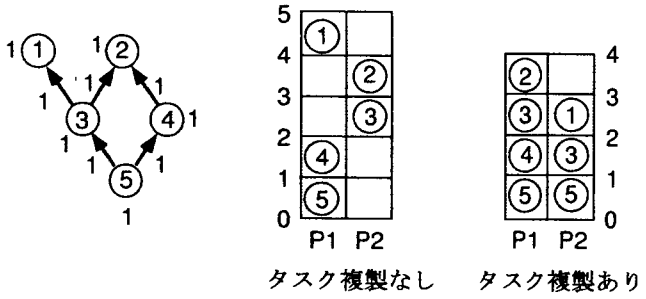


図 2.4: タスク複製の例 [4]

グループとする ($C_j \neq C_k$). $v \in C_j$ かつ $v \in C_k$ であるような節点 v を C_j と C_k の**共通節点**と呼ぶ. 複数の節点グループに属する節点は複製タスクに相当する. $v \in C_j$ かつ $u \in C_k$ かつ $u \notin C_j$ である (u, v) が E に存在するとき, かつそのときに限り C_j は C_k に**直接依存する**という. 以下の条件 C1 または C2 が満たされるとき, C_j は C_k に**依存する**という.

C1: C_j は C_k に直接依存する.

C2: C_j が C_i に直接依存し, かつ C_i が C_k に依存するような G の節点グループ C_i が存在する.

互いに依存のない節点グループの集合を**独立グループ集合**と呼ぶ. 図 2.5 のタスクグラフに対する独立グループ集合の例を図 2.6 に示す.

2.6 通信の一括化

分散メモリ型並列計算機における m バイトのメッセージの通信遅延時間は, $s + mb$ と近似できる [2, 21]. ここで s は**レイテンシ**と呼ばれる m に依存しない定数であり, b はネットワークの帯域幅の逆数である. レイテンシは, 送受信の準備のためのプロセッサの処理時間, またプロセッサを介して送信, 受信バッファを (OS のカーネル領域に) パッキング, アンパッキングする場合にはそれらの時間などを含む [21]. 通常レイテンシの値は 0 ではないので, できるだけ多くのメッセージを一度に転送する方が, 通信遅延を削減できる [3, 21]. このように複数のデータをまとめて 1 つのメッセージとして転送することを**通信の一括化**と呼ぶ. 通信の一括化は並列プログラムの最適化テクニックの 1 つとしてよく知られている [3]. 例として AP1000 の場合について, 通信遅延とメッセージ長

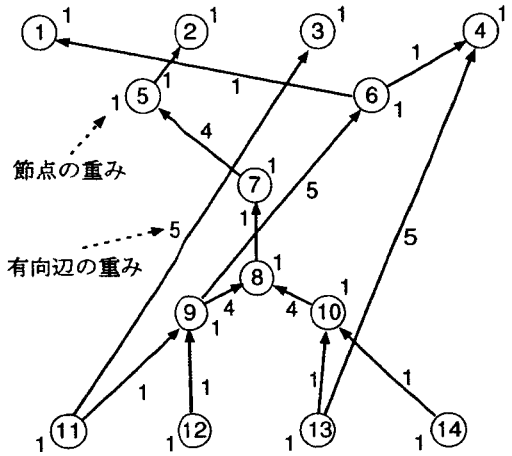


図 2.5: タスクグラフの例

の関係を図 2.7 に示す．２台のプロセッサ間で 32bit 整数を 1000 個転送する場合を考えると，図 2.7 から，整数１つを１メッセージとして合計 1000 個のメッセージを転送する場合は 3.32ms かかる．これに対して，整数 1000 個を１つのメッセージとして一括して転送する場合は 0.656ms しかかからない．すなわち整数１個のメッセージを 1000 回の通信で転送するよりも，整数 1000 個を１つのメッセージにまとめて，１回の通信で転送した方が約 200 倍速い．

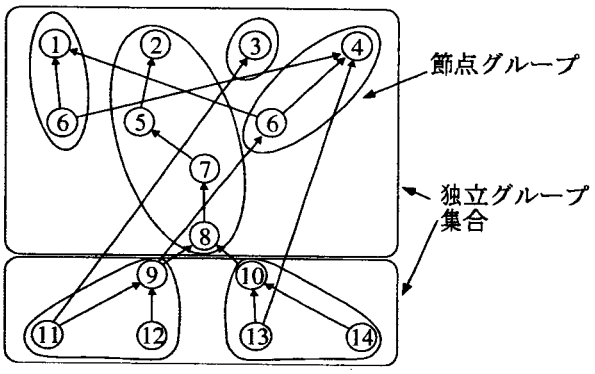


図 2.6: 独立グループ集合の例

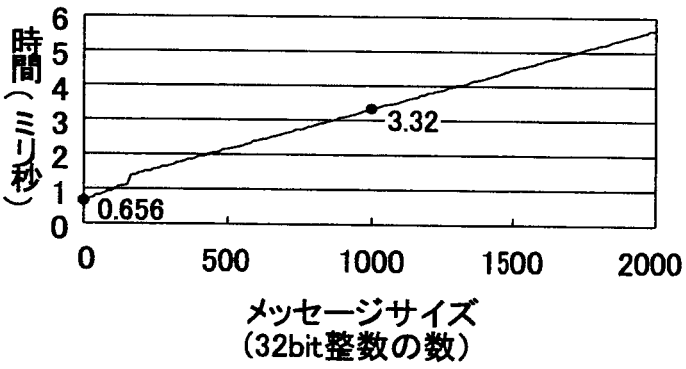


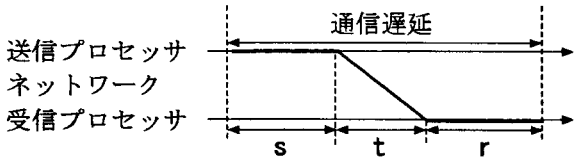
図 2.7: 通信遅延とメッセージ長の関係 (AP1000)

第3章　バルク同期スケジュールを生成するアルゴリズム

3.1　序言

分散メモリマシンを対象に細粒度タスクグラフのタスクスケジューリング問題を考える．分散メモリマシン上では，プロセッサの機械語命令1つの実行時間に対する通信遅延の比率は大きい．この通信遅延を支配しているのはソフトウェアオーバーヘッドである[14]．このソフトウェアオーバーヘッドに相当する処理の間は，プロセッサは他のいかなる計算も実行できない．このため細粒度並列プログラムでは通信と計算のオーバーラップは通信遅延をほとんど隠蔽できない．しかし2節で説明した伝統的なタスクスケジューリングモデル(以下，モデル)では，タスクの計算と通信は，通信がそのタスクの入力あるいは出力を伝えるものでない限り，完全にオーバーラップできると仮定している[4]．すなわち，モデルでは通信におけるソフトウェアオーバーヘッドはゼロあるいは無視できるものと仮定している．このため細粒度タスクグラフを対象とする場合，モデル上では理論的に速いスケジュールを生成する伝統的なタスクスケジューリングアルゴリズムを用いても，分散メモリマシン上で実際に速いスケジュールを生成できるとは限らない．これに対してメッセージベクトル化[3]などの通信の一括化(message packaging)を用いれば，ソフトウェアオーバーヘッドを削減できるため，分散メモリマシン上での並列プログラムの性能を著しく改善できることがよく知られている．

本章では，タスクスケジューリングアルゴリズムBCSH (Bulk Communication Scheduling Heuristic)を提案する．BCSHはバルク同期[20]スタイルのスケジュール（バルク同期スケジュール）を生成する．バルク同期スケジュールに基づく並列プログラムはバルク同期並列プログラム[20]となる．バルク同期並列プログラムでは通信を一括化しやすい[19]粗粒度並列プログラムであるので，BCSHは分散メモリマシン上で細粒度タスクグラフから実際に速いスケジュールを生成できる．バルク同期スケジュールを生成するタスクスケジューリングアルゴリズムはBCSHの他に存在しない．



s　送信のソフトウェアオーバーヘッド
t　メッセージ配送のハードウェアオーバーヘッド
r　受信のソフトウェアオーバーヘッド

図 3.1: 通信遅延のソフトウェアオーバーヘッド

LU分解 (ピボット選択なし), FFT, 連立方程式の反復解法のヤコビ法 (反復処理のボディのみ) について, BCSH が生成するスケジュール通りに動作する並列プログラムを生成し, その実行速度を分散メモリマシン AP1000[9] 上で評価した. その結果, BCSH に基づく並列プログラムでは, 利用するプロセッサ台数に応じて良好な速度向上が見られた.

3.2　通信の一括化とタスクスケジューリング

通信遅延は, 送信プロセッサがメッセージをネットワークに送出するためのソフトウェアオーバーヘッド s, ネットワークがそのメッセージを配送するためのハードウェアオーバーヘッド t, 受信プロセッサがネットワークからメッセージを受け取るためのソフトウェアオーバーヘッド r の3つの部分から成る (図 3.1). s と r を合わせて通信遅延のソフトウェアオーバーヘッドと呼ぶ. 本論文では, t はメッセージ長に依存するが s と r は依存しない実装 (ユーザーレベル通信など) を仮定する.

通信と計算をオーバーラップすれば, ハードウェアオーバーヘッドは隠蔽できるがソフトウェアオーバーヘッドは隠蔽できない. これは, ソフトウェアオーバーヘッドに相当する処理の間は, プロセッサは通信のためのソフトウェア処理を行い, 他の計算ができないためである. このためモデルと分散メモリマシンの間には以下に示すギャップが生じる. 図 3.2 を用いてこのギャップを説明する. モデル上では, タスク T_8 はタスク T_7 の実行終了後ただちに実行できる. さらに T_7 とタスク T_6 間の通信は, T_8 の計算と完全にオーバーラップできる. しかし分散メモリマシン上では, T_8 は T_7 の実行終了後すぐには実行できない. これは T_7 と T_6 の間の通信のソフトウェアオーバーヘッドのためである.

分散メモリマシン上では, プロセッサの機械語1命令の実行時間に対す

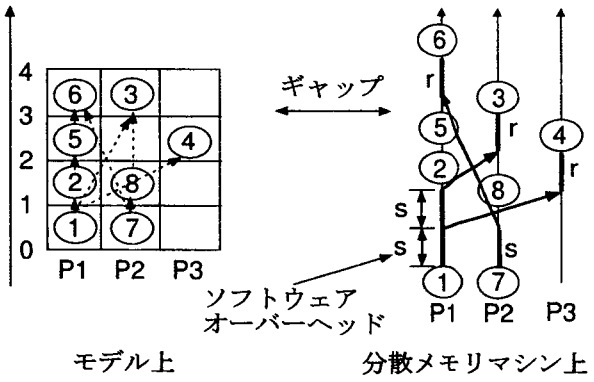


図 3.2: 伝統的なタスクスケジューリングモデルと分散メモリマシンの間のギャップ

る通信遅延のソフトウェアオーバーヘッドの比率は大きい．文献 [14] は，標準メッセージ通信ライブラリ MPI[12] のソフトウェアオーバーヘッドを最適化した実装におけるソフトウェアオーバーヘッドは，通信用コプロセッサを備えた並列計算機 AP1000[9] の場合，139μs から 260 μs であると報告している．3.6 節で評価実験に用いた細粒度タスクグラフにおけるタスクの平均実行時間は，AP1000 の場合，3μs から 9μs である．したがってソフトウェアオーバーヘッドの大きさは，細粒度タスク 15 個から 86 個分に相当する．すなわちモデルには，細粒度タスクグラフから生成した通信と計算のオーバーラップを含むスケジュールについては，分散メモリマシンをうまく近似できないという問題点がある．ソフトウェアオーバーヘッドの大きさを節点の重みに含めると悪い近似となる．この場合プロセッサ間の通信だけでなく，プロセッサ内の通信についてもソフトウェアオーバーヘッドが常に必要となるからである．

プロセッサ速度が速くなればソフトウェアオーバーヘッドは小さくなる．しかし，プロセッサ速度が速くなれば細粒度タスク 1 つの実行時間も小さくなる．このため将来プロセッサ速度が向上しても，この問題点の解決にはならない．この問題点の解決のためには，細粒度タスク 1 つの実行時間のソフトウェアオーバーヘッドの大きさに対する比率がゼロもしくは無視できる大きさにならなければならない．プロセッサ速度が向上するだけでは，この比率は小さくならない．

3.3　バルク同期スケジュール

3.2節で述べた問題点（通信のソフトウェアオーバーヘッドによる理論と現実のギャップ）を，モデルを拡張せずに回避する方法として，本研究ではバルク同期スケジュールの生成を提案する．本節ではバルク同期スケジュールを定義し，それが問題点の回避に有効であることを説明する．

タスクグラフ $G = (V, E, \lambda, \tau)$ のプロセッサ数 p のときのスケジュールを S とする．以下の条件 P1，P2 を満たすとき，またそのときに限り，$S'(\subseteq S)$ は S の**計算層**であるという．

P1: $q \neq q'$ かつ $(u,v) \in E$ となる3つ組 $\langle u,q,t\rangle$，$\langle v,q',t'\rangle$ が S' に存在するならば，$t'' \leq t' - \lambda(u)$ となる3つ組 $\langle u,q',t''\rangle$ が S' に存在する（必要な計算結果は自分で計算するので各プロセッサは S' の実行の際には通信の必要がない）

P2: $t_{min}(S') \leq t$ かつ $t + \lambda(v) \leq t_{max}(S')$ となる3つ組 $\langle v,q,t\rangle$ が S に存在するならば，3つ組 $\langle v,q,t\rangle$ が S' に存在する（S' は S の時間による分割の1要素である）．ここで $t_{min}(S') = \min\{t|(v,q,t) \in S'\}$，$t_{max}(S') = \max\{t + \lambda(v)|(v,q,t) \in S'\}$

S が以下の条件 P3 を満たす計算層の系列 $\langle S_1, S_2, \cdots, S_n\rangle$ に分割できるとき，またそのときに限り，S は**バルク同期スケジュール**であるという．

P3: 任意の $i(1 \leq i < n)$ について $t_{max}(S_i) + \tau_{suff}(S_i, S_{i+1}) \leq t_{min}(S_{i+1})$，ここで $\tau_{suff}(S_1, S_2) = \max\{\tau(u,v)|u \in S_1, v \in S_2\}$．ただし任意の u,v について $(u,v) \notin E$ の場合は便宜上 $\tau_{suff}(S_1, S_2) = \infty$ とみなす．

図2.5のタスクグラフに対するバルク同期スケジュールの例を図3.3に示す．

P1よりバルク同期スケジュールでは各計算層の計算はプロセッサ間で通信なしに行える．すなわち通信は計算層間でのみ必要となる．またP2より計算層間に時間的な重なりはない．このため各計算層の計算にともなうメッセージの送信 (受信) は計算層の計算の終了後 (開始前) まで遅らせる (早める) ことができる．そこで図3.4のように，各計算層を以下のように並列プログラム化する．

1. 計算を開始する前に，計算に必要なデータを可能な限り一括化されたメッセージとして各プロセッサから受信する．
2. すべてのデータを受信した後，スケジュール通りの順番で計算層の各タスクを実行する．

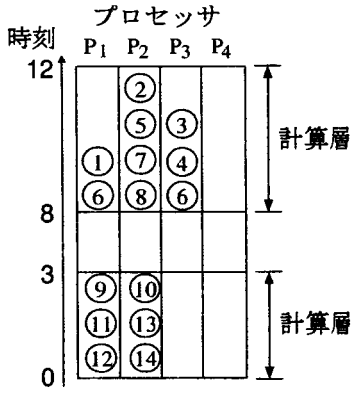


図 3.3: バルク同期スケジュールの例

3. 計算の終了後，計算結果のうち，他のプロセッサで必要とされるものを送信先プロセッサ毎に可能な限り一括化して送信する．

このようにプログラム化すれば，通信の一括化により通信のソフトウェアオーバーヘッドを（通信の一括化を行わない場合にくらべて）削減できる [19]．このため，3.2 節で述べたモデルの問題点を緩和できる．例えば図 3.5 は，3 つのメッセージの一括化により分散メモリマシン上での並列プログラムの実行時間が短くなる例である．通信を一括化するとメッセージ長が長くなるため通信のハードウェアオーバーヘッドは大きくなる（図 3.5 で $H > h$）が，通常 $(H - h)$ よりもソフトウェアオーバーヘッドの削減量の方が大きいため，通信の一括化は効果がある．P3 は，バルク同期スケジュールのメイクスパンが上のように通信を行う並列プログラムの実行時間を近似することをねらっている．上のようにプログラム化すれば，各計算層は粗粒度タスクになることに注意されたい．すなわち提案手法は細粒度タスクグラフから粗粒度並列プログラムを生成する．

3.4 実行時間が短いバルク同期スケジュールのもつ性質

分散メモリマシン用に並列プログラム化したときに実行時間が短いスケジュールは以下の 5 つの性質をもつと考えられる．

性質 1 バルク同期スケジュールであること（並列プログラム化するとき

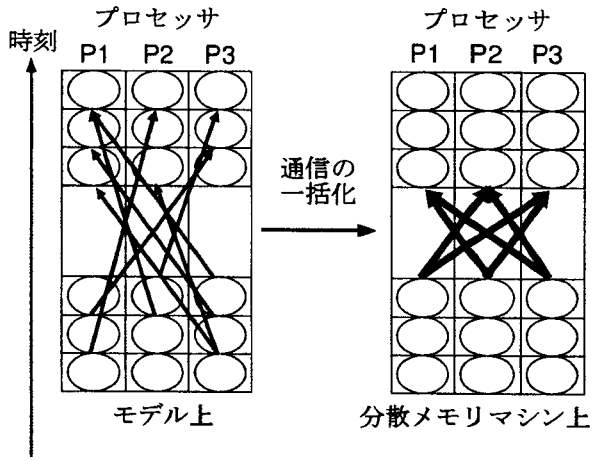


図 3.4: バルク同期スケジュールにおける通信の一括化

に通信の一括化を用いてプロセッサ間通信を実現することを仮定する）.

性質 2 各計算層において，プロセッサ間の計算の負荷バランスがよいこと.

性質 3 プロセッサ間の計算の負荷バランスがよい限り，各計算層はできるだけ長いこと.

性質 4 上の性質を満たすため以外に，複製タスクを含まないこと.

性質 5 メイクスパンが小さいこと.

性質 1 は 3.2 節と 3.3 節の考察の結果から得られる.

プロセッサ間の計算の負荷バランスが悪いとプロセッサがアイドルになる期間が現れる．最適スケジュールはプロセッサのアイドル期間を含まないとは限らないが，経験的には性質 3 と性質 4 を同時に満たすことを考えると，性質 2 は，分散メモリマシン用に並列プログラム化したときに実行時間が短いスケジュールのもつ性質の 1 つであると考える.

バルク同期スケジュールでは計算層が長いと通信の必要回数が減るので，通信を減らすという観点では計算層は長い方が望ましい．詳しくは 3.5 節で述べるが，長い計算層を作るためには，一般には複製タスクを多く作る必要がある．複製タスクには通信を減らすという利点があるが，同時に計算量が増すという欠点がある．すなわち，計算層の長さと複製タスク量の間にはトレードオフの関係がある．性質 3 と性質 4 は，トレー

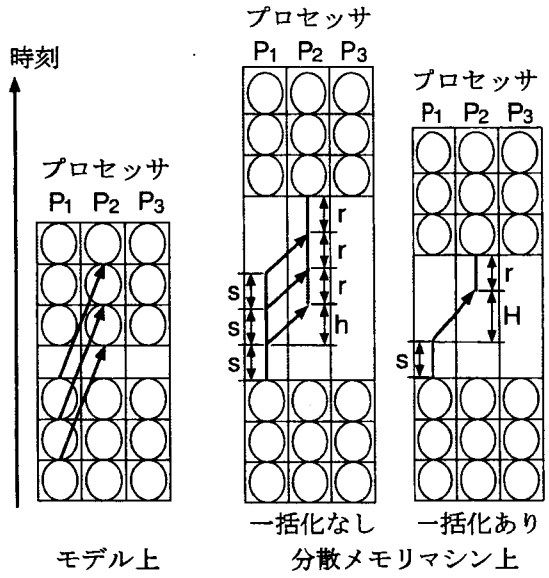


図 3.5: 通信の一括化によるソフトウェアオーバーヘッドの削減

ドオフの関係にある計算層の長さと複製タスク量のバランスのよいポイントを指摘している.

3.2 節と 3.3 節の考察の結果から, 単にメイクスパンが小さいだけでは, 分散メモリマシン用に並列プログラム化したときに実行時間が短いスケジュールとはならないことがわかる. しかしメイクスパンの値はまったく無視できるものではないことを性質 5 は指摘している. すなわち実行時間が短いバルク同期スケジュールとなるためには, 性質 1 から性質 4 を優先して満たした上で, 性質 5 を満たすべきである.

3.5 アルゴリズム BCSH

本節では 2 節で定義した独立グループ集合の概念を用いてバルク同期スケジュールを生成するアルゴリズム BCSH を示す. このために, まず 3.5.1 節でいくつかの記法を定義し, 次に 3.5.2 節で利用可能なプロセッサ台数と分散メモリマシンの通信特性を考慮した独立グループ集合のスケジューリングアルゴリズムを示す. 最後に 3.5.3 節で BCSH を示す.

3.5.1　記法

v のすべての直接後続節点の集合を $Succ(v)$ と書く．直接後続節点のない節点を**出力節点**と呼ぶ．**パスの長さ**を，そのパス上の節点の重みの和と定義する (有向辺の重みは加えない)．節点 u の**レベル** $level(u)$ は，u から出力節点への最長パスの長さである．レベル l_1 (l_1 から l_2) のすべての節点の集合を $V(l_1)$ ($V(l_1..l_2)$) と書く．G の節点のレベルの最大値を $l_{max}(G)$ と書く．G の有向辺の重みの最大値を $\tau_{max}(G)$ と書く．節点グループ C に属する節点の重みの総和 $\sum_{v\in C}\lambda(v)$ を $\omega(C)$ と書く．独立グループ集合 CC に属する節点の重みの総和 $\sum_{v\in(\cup_{C\in CC}C)}\lambda(v)$ を $W(CC)$ と書く．

3.5.2　独立グループ集合のスケジューリング

1つのタスクの計算結果を複数のプロセッサに送る場合，モデル上では各送信は並列に実行できる．これはソフトウェアオーバーヘッドをゼロと見なしているからである．しかし分散メモリマシン上では，これらの各送信は完全にはオーバーラップできない（図 3.2 の T_1 の送信参照）．このため（並列プログラム化したときに通信の一括化を行うことを前提として）スケジュールのメイクスパンが延びないならより少数のプロセッサを用いるべきである．例えば図 3.2 で T_4 を P3 でなく P2 にスケジュールして，T_1 から T_3 と T_4 への通信を一括化すれば T_1 の送信のソフトウェアオーバーヘッドを削減できる．この考えを**プロセッサ節約**と呼ぶ．p プロセッサが利用可能なとき，BCSH は独立グループ集合 CC を Graham のアルゴリズム LDSH（節点集合を節点の重みの総和の降順にもっとも負荷の軽いプロセッサに割り当てていく）[7] を用いて $P(\leq p)$ プロセッサにスケジュールする．ここで P は $LDSH(CC,P)=\min_{lb\leq x\leq p}LDSH(CC,x)$，$lb=\min(p,\lceil W(CC)/\max_{C\in CC}\omega(C)\rceil)$ （ただし $LDSH(CC,y)$ は CC を LDSH を用いて y プロセッサにスケジューリングしたときのスケジュールのメイクスパンである）を満たす最小の正整数である．以降では $LDSH(CC,P)$ の値を $LM(CC,p)$ と書く．一般に BCSH は，各計算層毎に異なる数のプロセッサを用いる．図 3.3 のスケジュールでは 4 プロセッサ中の 3 プロセッサしか使っていないことに注意されたい．

独立グループ集合 CC をスケジューリングするときのアルゴリズム LDSH の時間計算量は $O(|CC|\log|CC|)$ 時間である．ゆえに独立グループ集合 CC を p プロセッサにスケジューリングするときの本アルゴリズムの時間計算量は $O(p|CC|\log|CC|+N(CC))$ 時間である．ただし $N(CC)$ は重複して数えた CC に含まれる節点数 ($\sum_{C\in CC}|C|$) である．

3.5.3 独立グループ集合を用いたバルク同期スケジュールの構成法

以下の方法でバルク同期スケジュールを生成できる.

1. 節点のレベルをいくつか選んで，タスクグラフ G で選んだレベルで部分グラフに分割し，(必要ならば複製タスクを用いて) 各部分グラフから独立グループ集合を構成する．例えば図 2.6 は，図 2.5 のタスクグラフをレベル 4 と 5 の間で分割し，タスク T_6 を複製して構成した 2 つの独立グループ集合を表している.
2. 各独立グループ集合が 1 つの計算層に対応するように，節点グループ単位で節点をプロセッサに割り当ててバルク同期スケジュールを構成する．例えば図 3.3 は，図 2.6 の独立グループ集合に基づいて構成されたバルク同期スケジュールを表している.

この方法によってメイクスパンの小さいバルク同期スケジュールを生成するためには，タスクグラフをどのレベルで分割するか，独立グループ集合をどのように構成するかをうまく決定する必要がある．BCSH は以下の手順でバルク同期スケジュールを生成する.

l を任意の節点のレベルとする．同じレベルの節点は互いに依存しない．ゆえに $V(l)$ から独立グループ集合 $\cup_{v \in V(l)}\{\{v\}\}$ を構成できる．$V(l..l+k)(k$ は非負正数) から独立グループ集合 CC を構成されているとする．このとき $V(l..l+k+1)$ から独立グループ集合を作ることを考える．u をレベル $(l+k+1)$ の任意の節点とする．節点 u を複製して u の直接後続節点が属する節点グループのすべてに u を追加すれば (この処理をタスクの全複製と呼ぶ)，$V(l..l+k) \cup \{u\}$ から独立グループ集合 $(\{C \cup \{u\} | C \in CC, C \cap Succ(u) \neq \emptyset\}) \cup (\{C | C \in CC, C \cap Succ(u) = \emptyset\})$ を作ることができる (図 3.6 参照)．CC をこの独立グループ集合として再定義する．$V(l+k+1)$ に属する各節点に対してこの複製を繰り返せば，$V(l..l+k+1)$ から独立グループ集合を作ることができる.

さらに，共通節点をもつ 2 つの節点グループをマージすれば，複製を減らすことができて独立グループ集合を改良できる可能性がある．1 つの独立グループ集合に属し，かつ共通節点をもつ 2 つの節点グループ C_i と C_j をマージすれば，タスク複製を減らすことができる．しかしマージすれば，C_i と C_j 間の並列性が失われる．マージしてできる節点集合の大きさが $LM(CC,p)$ 以上の場合，BCSH は節点集合 C_i と C_j をマージしない．BCSH はこの場合にはマージによりメイクスパンが延びる可能性が高いとみなす．BCSH はマージしてできる節点集合ができるだけ大きくならない順番で C_i と C_j を選ぶ．これはマージしてできる節点集合が

他の節点集合より極端に大きくなりスケジュールの負荷バランスが悪くなることを避けるためである．

このようにBCSHは，節点を複製したり，節点グループどうしをマージして節点グループを成長させながら，出力節点から入力節点へ向かってレベル毎に節点をグループ化することにより，並列に計算できる連結成分を見つける．レベル $(l+k+1)$ が $l_{max}(G)$ に達するまでの間，$V(l..l+k+1)$ から生成した独立グループ集合に属する節点グループ間の負荷バランスが崩れない場合，BCSHは $V(l..l+k+1)$ から生成した独立グループ集合を新しいグループ層として確定し，アルゴリズムを終了する．途中で負荷バランスが崩れた場合，BCSHは $V(l..l+k)$ から生成した独立グループ集合を新しいグループ層として確定し，上記の処理をレベル $(l+k+1)$ から再開する．成長した節点グループは小さいままの節点グループよりも直接先行タスクが比較的多い可能性が高いので，一度成長した節点グループはますます成長しやすく，小さいまま節点グループは成長しにくい．すなわち独立グループ集合のバランスは，崩れ出すとレベルを進めるにつれ，ますます崩れやすい．BCSHはバランスが崩れたまま独立グループ集合を成長させるよりは，最後にバランスがよかったレベル $(l+k+1)$ から処理をやりなおす方がよいとみなすことを意味する．

BCSHは負荷バランスが崩れているかいないかを以下のように判断する．独立グループ集合 CC の全タスクを完全に並列に計算できるなら，その計算時間は $W(CC)/p$ となる．条件 $LM(CC,p) \leq W(CC)/p + \tau_{max}(G)$ が成り立つとき，かつそのときに限りBCSHは CC の負荷バランスがよいと判断する．すなわちBCSHは理想的にバランスがとれている状態から $\tau_{max}(G)$ までのバランスの崩れは容認する．これはバランスの崩れを解消するために通信層を入れると確実に $\tau_{max}(G)$ の遅延がかかるためである．

以下にアルゴリズムBCSHを示す．

アルゴリズムBCSH
入力：タスクグラフ $G=(V,E,\lambda,\tau)$ と利用可能なプロセッサ数 p
出力：G の p に対するバルク同期スケジュール S
begin
V の各節点のレベルを計算する;
$\tau_{max}(G)$ を計算する;
$l := 1$;
{ 出力節点側から1つずつ負荷バランスのよい
計算層を見つける }
while $l \leq l_{max}(G)$ **do begin**

$L := \cup_{v \in V(l)}\{\{v\}\}$; {$L$ は独立グループ集合 }
balanced := **true**; $k := 1$
{L の負荷バランスがよい限り，L を成長させる }
while $l + k \leq l_{max}(G)$ **and** *balanced* **do begin**
CC:=L;
{$V(l..l+k-1)$ から構成した独立グループ集合
L をもとに $V(l..l+k)$ から構成した
独立グループ集合を CC とする }
{ タスク複製により節点グループを成長させる }
foreach $u \in V(l+k)$ **do**
foreach $C \in CC$ **do**
if $C \cap Succ(u) \neq \emptyset$ **then**
$C := C \cup \{u\}$;
{ 節点グループのマージにより
CC 中の余分な複製タスクを削減する }
LM:=LM(CC,p);
CC の要素を節点の重みの総和の昇順に
ソートしたものを $\langle C_1, C_2, \cdots, C_n \rangle$ とする;
$i := 1$;
{C_i を昇順に選ぶ }
while $(i < n)$ **do begin**
$(CC - \{C_i\})$ の要素を C_i との共通節点の
重みの総和の降順にソートしたものを
$\langle D_1, D_2, \cdots, D_m \rangle$ とする;
$j := 1$;
{D_j を降順に選ぶ }
while $(j \leq m)$ **do begin**
if $\omega(D_j \cup C_i) < LM$ **then begin**
$D_j := D_j \cup C_i; CC := CC - C_i$;
end; $j := j + 1$
end; $i := i + 1$
end
$t := \omega(V(l+1..l_{max}(G)))$;
if $t \geq t/p + \tau_{max}(G)$ **then begin**
{$V(l+1..l_{max}(G))$ に少ししか節点が
ないときはタスクグラフを分割しない }
balanced := $(LM(CC,p) \leq W(CC)/p + \tau_{max}(G))$;

```
      if balanced then
        begin L := CC; k := k + 1; end
    end;
  end;
  { 最後にバランスがよかった L を計算層とする }
  3.5.2 節のアルゴリズムを用いて
  L をスケジュールする; l := l + k;
end; {while l ≤ l_max(G) do begin}
end
```

以下ではBCSHの時間計算量が$O(|V|^4)$であることを示す.

各タスクのレベルは$O(|V|+|E|)$時間で計算できる. $\tau_{max}(G)$は$O(|E|)$時間で計算できる. BCSHはタスクグラフをレベル毎に走査する. BCSHは計算層の候補Lのバランスがよい間, 走査するレベルを進め, 走査中にLのバランスが悪くなった場合のみ1レベルだけ後戻りする. ゆえにBCSHは高々2回しか各レベルを走査しない.

BCSHが各レベルに対して行う処理は, タスクの全複製, 節点グループのマージ, 計算層の候補Lのバランス判定の3つである. レベルlを走査しているときの計算層の候補CCを$CC(l)$と書く. レベルlにおける各処理の時間計算量は以下の通りである.

各節点が属する節点グループのリストを管理するようにすれば, 1つのタスクの全複製は$O(|V|^2)$時間で計算できる. $|V(l)|$個のタスクの全複製を行うので, タスクの全複製は$O(|V|^3)$時間で計算できる.

$|CC(l)|$は高々$|V|$である. ゆえにCCのソートは$O(|V|\log|V|)$時間で計算できる. CCに含まれる各節点グループ間の共通節点の重みの総和をテーブルとして管理し, タスクuの全複製を行うときに, このテーブルを更新するようにすれば, $(CC-\{C_i\})$のソートは$O(|V|\log|V|)$時間で計算できる. また, 節点グループに属する節点のリストを節点の番号の昇順リストとして管理すれば, 2つの節点グループのマージ処理は$O(|V|)$時間で計算できる. ゆえに全節点グループのマージ処理は, $O(|V|^3)$時間で計算できる.

$|V|$個より多くのプロセッサを用いる必要はないので, 一般性を失うことなく$p \leq |V|$と仮定できる. また$|CC(l)|$は高々$|V|$であるので, 複製タスクを重複して数える場合のCCに含まれる節点の数は高々$|V|^2$個である. ゆえに3.5.2節より$LM(CC,p)$は$O(|V|^2\log|V|)$時間で計算できる. CCの節点グループに属する節点の番号を列挙した配列を節点の番号でソートし, 重複した節点番号を読みとばしがらソート結果を走査す

れば，$W(CC)$ は $O(|V|^2 \log |V|^2)$ 時間で計算できる．ゆえに計算層の候補 L のバランス判定は $O(|V|^2 \log |V|^2)$ 時間で計算できる．

$|CC(l)|$ は高々 $|V|$ であるので，最後にバランスがよかった計算層の候補 L に含まれる節点グループの数は高々 $|V|$ 個である．ゆえに複製タスクを重複して数える場合の L に含まれる節点の数は高々 $|V|^2$ 個である．ゆえに 3.5.2 節より L のスケジューリングの時間計算量は $O(|V|^2 \log |V|)$ 時間である．

以上をまとめると，$|E| \leq |V|^2, l_{max}(G) \leq |V|$ より，BCSH の時間計算量は $O(|V| + |E|) + \sum_{1 \leq l \leq l_{max}(G)} 2 \times O(|V|^3) = O(|V|^4)$ である．

3.6 評価実験

3.6.1 実験に用いたタスクグラフ

既存の研究でアルゴリズムの評価に用いるタスクグラフは，プログラムとして意味のないランダムグラフが多かった．また実際のプログラムの計算を表すタスクグラフを用いている場合でも，節点数は小さかった．本論文では，以下に示す方法で逐次プログラムからタスクグラフを機械的に生成することにより，実験には実際のプログラムの計算を表すタスクグラフ（節点数数十万規模）を用いた．

1. もとの逐次プログラム SC にデータ依存関係に関する情報を実行トレース情報として生成するコードを挿入する．コードを挿入されたプログラムを ASC と呼ぶ．
2. ASC を逐次マシン上で実行する．ASC は，SC に含まれるすべての計算を実行すると同時に，SC 中のすべての代入文インスタンス間のフロー依存を（静的にではなく）動的に検出する．

上記の方法で，逐次 C 言語プログラム SC から，SC 中の代入文の各インスタンスとその間のフロー依存関係をそれぞれ節点と有向辺とするタスクグラフ G を生成する．SC 中のすべての文の平均実行時間を単位時間とする．各節点の重みを先の平均実行時間で近似する．BCSH については，各有向辺の重みは 1 ワードのデータの全対全通信の実行にかかる通信遅延時間とする．これは生成される並列プログラムでは，すべての通信は一括化されたメッセージの全対全通信 (`MPI_Alltoallv()`) として実現されるからである．一括化されたメッセージの長さは 1 ワードより長くなり，そのメッセージ長はメッセージ毎に異なるが，本論文では，それぞれの全対全通信の通信遅延時間を 1 ワードのデータの全対全通信の

通信遅延時間で近似する．図3.10のプログラムのメッセージ長は，16プロセッサの場合，最小32ワード，平均123ワード，最大1138ワードである．16プロセッサのAP1000上でのmワードメッセージの全対全通信にかかる時間は$(0.02m + 4.5)$ミリ秒であり，mの影響は比較的小さい．例えば1ワードと123ワードの全対全通信時間の比率は約1.54である．DSHについては，各有向辺の重みは1ワードのデータの1対1通信の実行にかかる通信遅延時間とする．これはDSHが生成したスケジュールには通信の一括化ができるという保証はないため，DSHを用いて生成する並列プログラムでは，すべての通信は1ワードメッセージの1対1通信(`MPI_Send()`, `MPI_Recv()`)として実現されるからである．

3.6.2　実験に用いた分散メモリマシン

実験には富士通AP1000を用いた．AP1000は16から1024台のSPARCプロセッサ(クロック25MHz, 15MIPS, 8.33MFLOPS)を2次元トーラスネットワークで接続した分散メモリマシンである．各プロセッサは16MBの主記憶と128KBのキャッシュメモリと通信用のコプロセッサを備えている．さらにAP1000は，2次元トーラスネットワークの他に，ブロードキャスト専用のバスと，バリア同期専用の木構造のネットワークを備えている．ネットワークの帯域幅は，トーラスが25MB/s，バスが50MB/sである．バリア同期の要求から成立までの時間は$1.6\mu s$である．

3.6.3　タスクスケジューリングアルゴリズムの評価基準

入力がタスクグラフGとプロセッサ数pのときのタスクスケジューリングアルゴリズムAの速度向上率を2種類定義する．

Aが生成したスケジュールのメイクスパンを$M_p(G)$とする．Gの全タスクの重みの総和を$M^*(G)$とする．このとき$S_M(p) = M^*(G)/M_p(G)$を，入力がタスクグラフGとプロセッサ数pのときのタスクスケジューリングアルゴリズムAの**メイクスパン速度向上率**と定義する．Gの全タスクの重みの総和は，プロセッサ数1のときのGの自明な最適スケジュールのメイクスパンである．ゆえにメイクスパン速度向上率はモデル上での速度向上率を表す．

Aが生成したスケジュールに基づく並列プログラムの実行時間を$T_p(G)$とする．このとき$S_P(p) = T_1(G)/T_p(G)$を，Aのpに対する**プログラム速度向上率**と定義する．$S_P(p)$は，Aを用いて得られる並列プログラムの分散メモリマシン上での速度向上率を表す．

表 3.1: スケジュールの特徴（ヤコビ 128）

p	$\tau_{max}(G)$	複製率	計算層数	メッセージの平均長
2	200	1.014930	2	64
4	300	1.014930	2	32
8	480	1.014930	2	16
16	870	1.029850	2	8
32	1670	1.074640	2	4

3.6.4 通信の一括化の効果

AP1000 上で通信の一括化を行う場合と行わない場合のそれぞれについて BCSH により得られる速度向上率を図 3.7 に示す．図 3.7 の例では，スケジュールが同じであっても通信の一括化により速度向上率が 2 倍から 8 倍改善されている．

図 3.7 の並列プログラムのもととなったタスクグラフ $G = (V, E, \lambda, \tau)$ とスケジュール S の特徴を表 3.1 に示す．表 3.1 の**複製率**は，スケジュール中の全タスクインスタンスの重みの総和のタスクグラフ中の全タスクの重みの総和に対する比率 $(\sum_{(v,q,t)\in S} \lambda(v) / \sum_{v\in V} \lambda(v))$ である．複製率が 1 の場合は，そのスケジュール中には複製タスクは存在しない．表 3.1 の**計算層数**は，スケジュール中の計算層の数を表す．表 3.1 の**メッセージの平均長**は，一括化されたメッセージの長さ（ワード単位）の平均値を表す．ヤコビ 128 に対して BCSH は，計算層が 2 つだけのスケジュールを生成する．ゆえにこのスケジュールに基づく並列プログラムは，全対全通信をただ 1 回だけ行うプログラムとなる．利用可能プロセッサ数 p が大きくなれば，通信の一括化の効果はうすれる．これは問題サイズが固定されているので，p が大きくなれば一括化されたメッセージの平均長は短くなるからである．

3.6.5 プロセッサ節約の効果

プロセッサ節約を行う場合と行わない場合のそれぞれについて BCSH に基づく並列プログラムの実行時間を図 3.8 に示す．またプロセッサ節約を行う場合と行わない場合のそれぞれについて BCSH が生成したスケジュールの特徴を表 3.2 と表 3.3 に示す．利用可能なプロセッサ数が比較的小さい場合は，プロセッサ節約の効果はない．これは節約するほどプロセッサ数が利用可能でないためである．しかし，利用可能なプロセッサ数が大きくなれば，プロセッサ節約の効果も大きくなる．例えば図 3.8

表 3.2: スケジュールの特徴 (LU 分解 128, プロセッサ節約あり)

p	$\tau_{max}(G)$	複製率	計算層数	メッセージの平均長
2	120	1.007860	24	616
4	250	1.013800	20	231
8	500	1.053620	20	159
16	1070	1.152990	19	121
32	2270	1.285810	15	96

表 3.3: スケジュールの特徴 (LU 分解 128, プロセッサ節約なし)

p	$\tau_{max}(G)$	複製率	計算層数	メッセージの平均長
2	120	1.007860	24	394
4	250	1.013800	20	259
8	500	1.053620	20	128
16	1070	1.152990	19	116
32	2270	1.285810	15	87

では, 32 台のプロセッサが利用可能な場合, プロセッサ節約によりプログラムの実行時間は半分に減っている.

3.6.6　DSH との比較

LU 分解 (問題サイズ 32, ピボット選択なし) を AP1000 で解いた場合の BCSH と DSH のメイクスパン速度向上率と速度向上率を図 3.9 に示す. また BCSH が生成したスケジュールの特徴を表 3.6 に示す. 図 3.9 では, DSH を用いて生成された並列プログラムは, 対応する逐次プログラムよりもかなり遅い. これに対して, BCSH を用いて生成された並列プログラムは, 対応する逐次プログラムよりも速く, かつ, DSH を用いて生成された並列プログラムよりも 7 倍から 23 倍速い. BCSH を用いる場合でも, 通信の一括化を行わないと並列化の効果はない. 表 3.6 より BCSH では平均 10 から 125 ワードのメッセージを一括化していることがわかる.

3.6.7　BCSH を用いて生成した MPI 並列プログラムの性能

LU 分解 (問題サイズ 128, ピボット選択なし), 高速フーリエ変換 (問題サイズ 8192), ヤコビ法 (問題サイズ 512, 反復処理のボディのみ) の場合について, MPI を用いた C 言語並列プログラムの速度向上率を図 3.10

表 3.4: プロセッサ節約による各計算層における使用プロセッサ数の変化 (LU 分解 128, $p = 32$)

計算層	プロセッサ節約なし	プロセッサ節約あり
14(出力側)	18	18
13	18	18
12	20	20
11	18	18
10	19	19
9	21	21
8	23	**20**
7	21	21
6	23	23
5	23	**21**
4	20	**19**
3	19	**18**
2	15	15
1	20	20
0(入力側)	1	1

に示す. 図 3.10 では, BCSH は良好な速度向上率を示している. 各スケジュールの特徴を表 3.3 から 3.8 に示す. プロセッサ数が 8 の場合について, LU 128 のスケジュールの詳細を表 3.9 に示す. 表 3.9 は, LU128 のスケジュールで各計算層で各プロセッサに割り当てられたタスクの重みの総和を示している. 重みの総和 0 は, そのプロセッサがその計算層で割り当てられたタスクがないことを意味する. 表 3.9 から, プロセッサ節約により BCSH は必ずしも利用可能なプロセッサのすべてを使うわけではないことがわかる. 例えば計算層 16 では 8 プロセッサのうち 7 プロセッサしか使われていない. これらの結果から以下のことがわかる.

- 高速フーリエ変換に対して, BCSH はタスク複製をうまく用いて通信を一切含まないスケジュールを生成している.
- ヤコビ法のタスクグラフは, 長くて独立な計算の後に短くて互いに依存した計算が続く. ヤコビ法に対しては, BCSH は全対全通信を 1 回だけ挿入するポイントを適切に見つけている.
- LU 分解に対しては, 利用可能なプロセッサ数が問題サイズに対して適切である場合, BCSH はタスク複製をうまく用い, かつ, 全対全通信を挿入するポイントを適切に見つけている. これは利用可能なプロセッサ数が 4 台のとき得られる速度向上率が理想値 4 に近いことからわかる.

表 3.5: プロセッサ節約による各計算層間における平均メッセージ長の変化（LU 分解 128, $p = 32$）

計算層	プロセッサ節約なし	プロセッサ節約あり
14-13 (出力側)	62	62
13-12	63	65
12-11	68	76
11-10	93	103
10-9	102	114
9-8	91	123
8-7	143	143
7-6	134	136
6-5	116	141
5-4	106	112
4-3	105	111
3-2	69	82
2-1	60	62
1-0 (入力側)	12	14

表 3.6: スケジュールの特徴（LU 分解 32）

p	$\tau_{max}(G)$	複製率	計算層数	メッセージの平均長
2	120	1.107590	5	125
4	250	1.303610	5	38
8	500	2.354660	4	29
16	1070	6.640300	2	10

表 3.7: スケジュールの特徴（高速フーリエ変換 8k）

p	$\tau_{max}(G)$	複製率	計算層数	メッセージの平均長
2	330	1.025640	1	-
4	480	1.153850	1	-
8	800	1.487180	1	-
16	1400	2.230770	1	-
32	2700	3.794870	1	-

計算層が 1 つしかないので通信を含まないスケジュールとなっている

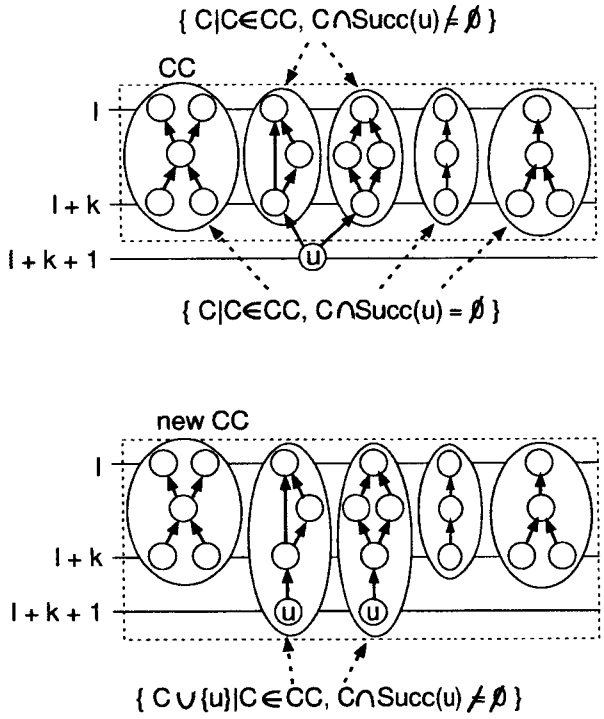


図 3.6: 並列計算できる連結成分の抽出

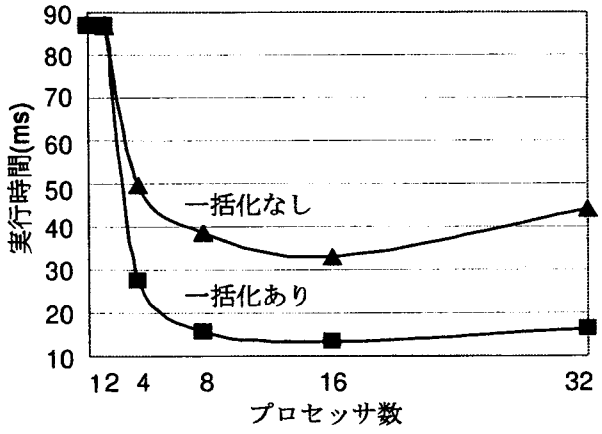


図 3.7: 通信の一括化の効果 (問題サイズ 128 のヤコビ法を AP1000 で解く場合)

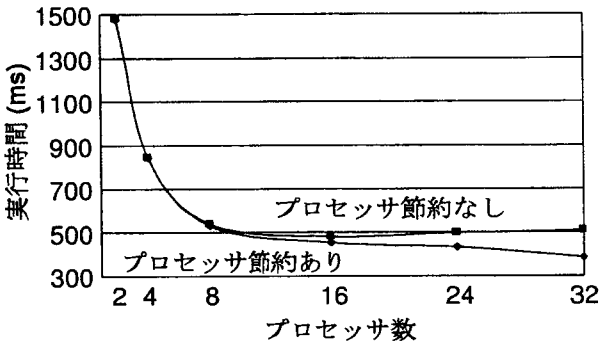


図 3.8: プロセッサ節約の効果 (問題サイズ 128 の LU 分解を AP1000 で解く場合)

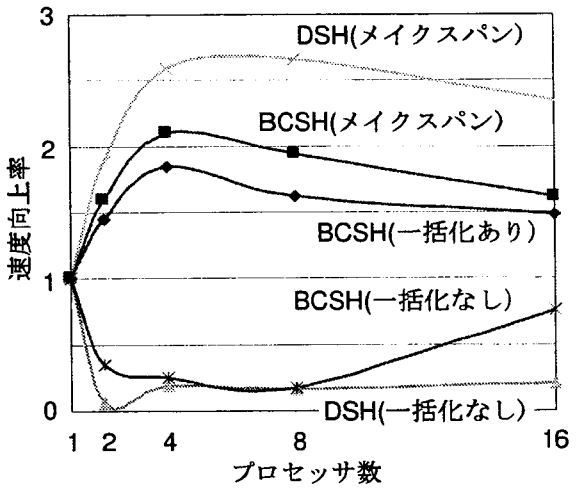


図 3.9: DSH と BCSH の比較 (問題サイズ 32 の LU 分解を AP1000 で解く場合)

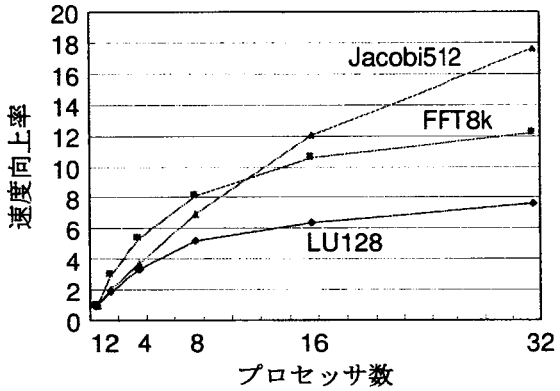


図 3.10: AP1000 上での速度向上率

表 3.8: スケジュールの特徴（ヤコビ 512）

p	$\tau_{max}(G)$	複製率	計算層数	メッセージの平均長
2	200	1.994220	1	-
4	300	1	2	128
8	300	1	2	64
16	870	1	2	32
32	1670	1	2	16

$p = 2$ の場合，計算層が 1 つしかないので通信を含まないスケジュールとなっている

表 3.9: 各計算層の計算負荷 (LU 分解 128, $p = 8$)

計算層	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
19（出力側）	294	294	294	244	243	242	242	242
18	1713	2296	1713	2396	1784	1876	1835	1863
17	1239	1191	1149	1551	1149	1356	1149	1372
16	2799	2668	2689	2884	2875	2835	0	2667
15	3019	3139	2898	3209	3186	3208	758	2959
14	3123	3258	3124	3613	3132	3260	3123	3160
13	3834	3469	3721	3875	3605	3483	3469	3946
12	5247	4953	5263	5659	4949	5277	4950	5137
9	10332	10125	10180	10340	10107	10216	10107	10133
11	6958	6974	6860	7470	7007	6861	6985	6981
10	9188	9138	9220	9355	9274	9335	9122	9123
8	13554	13144	13599	13703	13482	13264	13369	12889
7	6935	6505	6515	6545	6485	6495	6525	6535
6	9801	9234	9347	9234	9234	9234	9234	9234
5	5978	5400	5598	5355	5385	5430	5370	5415
4	4053	3402	3955	3771	3402	3402	3402	3402
3	2758	2750	2772	2758	2744	0	2744	2753
2	1508	1667	1760	1521	1547	0	1427	1534
1	2743	2938	3146	2759	2485	2150	2858	2516
0（入力側）	0	0	0	0	0	0	294	0

第4章　完全2分木のスケジューリングアルゴリズム

4.1　序言

TSP_τ でメイクスパン最小のスケジュールを求める問題はNP困難である．とくに，$\tau = 0$ かつ $p = 2$ に限定した場合や，p を任意に設定でき，かつ，すべてのタスクの処理時間が単位時間であると限定した場合 [17] でもNP困難である．このため様々な近似アルゴリズムが考えられている．既知の近似アルゴリズムと本稿で提案する近似アルゴリズムについてまとめたものを表4.1に示す．ここで，n は G のノード数，a は G のアーク数を表す．

応用例の多い，すべてのタスクの処理時間が単位時間である完全2分木に対してアルゴリズムPY[17]を適用し，冗長な再計算を省略すれば，木を根から高さ $\lfloor \log(\tau+2) \rfloor$ 段毎に切るというスケジュールが得られる．このスケジュール法は単純だが，どのプロセッサも通信待ちのためタスクを処理できない τ 時間が周期的に現れる．例えば高さ4の完全2分木に対して $\tau = 2$ のとき，直接には図4.1のようなスケジュールが得られるが，冗長な再計算を省略すれば，図4.2のスケジュールが得られる．ここで，例えば図4.2では，タスク6はプロセッサ P_3 により時刻2に処理

表 4.1: TSP_τ の最適解の近似解を求めるアルゴリズム (n, a はそれぞれDAGのノード数とアーク数)

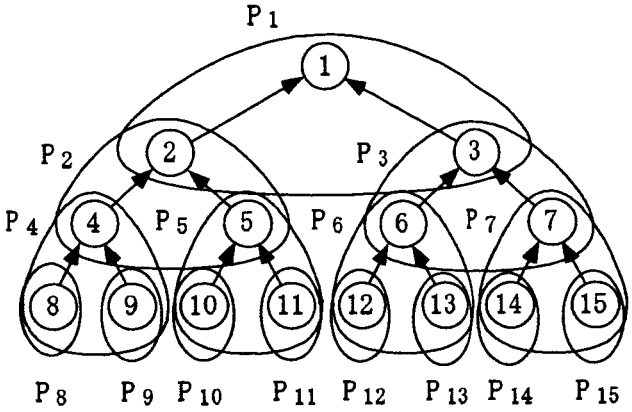
Algo.	G[7]	PY[17]	ALG1	ALG2
DAGの形状	任意	任意	完全2分木	完全2分木
タスクの処理時間	任意	任意	すべて等しい	すべて等しい
通信遅延 τ	0	任意	任意	任意
プロセッサ数	任意数 p	n	高々 $\frac{n+1}{2}$	高々 $\frac{n+1}{2}$
近似精度	高々 $2-\frac{1}{p}$	高々2	平均1.1から1.3	平均1.1から1.4
計算量	$O(n)$	$O(n(a+n\log n))$	$O(n)$	$O(n)$

されることなどを表す．プロセッサ P_1 がタスク 3 を処理するにはタスク 6 の結果が必要である．$\tau = 2$ であるので時刻 3，4 ではタスク 6 の結果を P_2 から P_1 へ送信するために消費されていて，その間 P_1 はアイドリングしている．

結合則をみたす 2 項演算子 $\circ$ に対して式 $a_1 \circ a_2 \circ \cdots \circ a_m$ を評価する問題では，通信遅延を無視すれば完全 2 分木状に計算するのが最適解の 1 つとなる．この問題では，もとの完全 2 分木をどのタスクにおいても通信待ちがないスケジュールが得られる 2 分木（以降変形木）に変形するという方法（以降アルゴリズム NKF）が知られている [13]．図 4.3 は $\tau = 2$ の場合に高さ 7 の完全 2 分木を変形して得られる，式 $a_1 \circ a_2 \circ \cdots \circ a_{16}$ を計算する変形木である．同様のタスクを考えると図 4.2 と図 4.7 の T_4 の DAG で式 $a_1 \circ a_2 \circ \cdots \circ a_{16}$ を計算できる．図 4.3 のスケジュールでは，必要な結果を得るまでの通信時間の間にプロセッサをアイドリングさせずに，他のタスク処理を行なわせている．例えばプロセッサ P_1 は根のタスクの実行において，プロセッサ P_4 の計算結果を通信により受けとる必要があるが，このとき通信待ちは起こらない．対象をこのような問題に限れば，完全 2 分木の変形が認められるが，本稿では，一般に DAG は並列アルゴリズムそのものを表すので，変形はできないと考えている．

本稿では，プロセッサが必要なだけ使える場合の，処理時間が単位時間のタスクだけからなる完全 2 分木を対象とした通信遅延を考慮したスケジューリングアルゴリズム ALG1 と ALG2 を示す．さらに DAG のスケジュールのメイクスパンの既知の下界の改良を行ない，ALG1，ALG2，PY の解のメイクスパンの近似精度をより正確に評価する（図 4.4）．この図は，各 τ の値に対して，高さ 1 から 20 までの完全 2 分木に対するアルゴリズムの解のメイクスパンと改良した下界に対する比の平均値を表している．高さ 20 の完全 2 分木のノード数は 100 万を越える．ALG1，ALG2 の両方とも必要なプロセッサ数は高々 n 台で，再計算は行なわない．ALG1，ALG2 は，それぞれ最適解の平均高々1.1 倍から 1.3 倍程度，1.1 倍から 1.4 倍程度のメイクスパンで走るスケジュールを得る．これに対して PY の解は平均 1.5 倍程度以上かかっている．また ALG1 と同じ数のデータを処理できる変形木とのメイクスパンの比率を計算することにより，変形を行なわない場合のオーバーヘッドは，ほとんどの場合，平均高々1.15 倍以内であることが示せる．

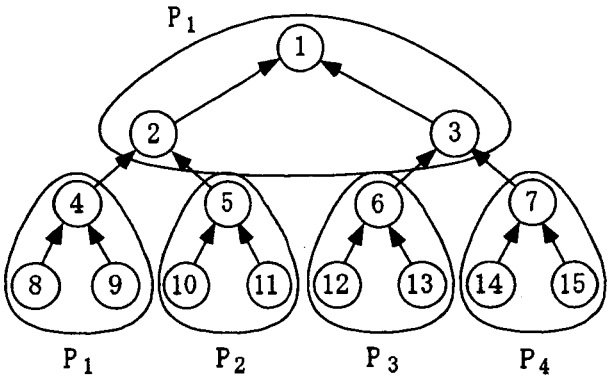
ALG1 の解のメイクスパンは最適解のそれに近いことがわかるが，プロセッサ割当が複雑となる．これに対して ALG2 は，ALG1 にはやや劣るが，やはり最適解に近いメイクスパンのスケジュールが得られ，かつ，必要なプロセッサ台数も ALG1，PY よりも少ない．さらにプロセッサ割



P_1～P_{15}の15台のプロセッサに割り当てる

プロセッサ＼時刻	0	1	2	3	4	5	6	7	8	9	10
P_1									3	2	1
P_2					5	4	2				
P_3					7	6	3				
P_4			9	8	4						
P_5			11	10	5						
P_6			13	12	6						
P_7			15	14	7						
P_8	8										
P_9	9										
P_{10}	10										
P_{11}	11										
P_{12}	12										
P_{13}	13										
P_{14}	14										
P_{15}	15										

図 4.1: アルゴリズム PY のスケジュールの例 ($\tau = 2$)



$P_1 \sim P_4$ の4台のプロセッサに割り当てる

プロセッサ＼時刻	0	1	2	3	4	5	6	7
P_1	9	8	4			3	2	1
P_2	11	10	5					
P_3	13	12	6					
P_4	15	14	7					

図 4.2: 再計算を省いたアルゴリズム PY のスケジュールの例 ($\tau = 2$)

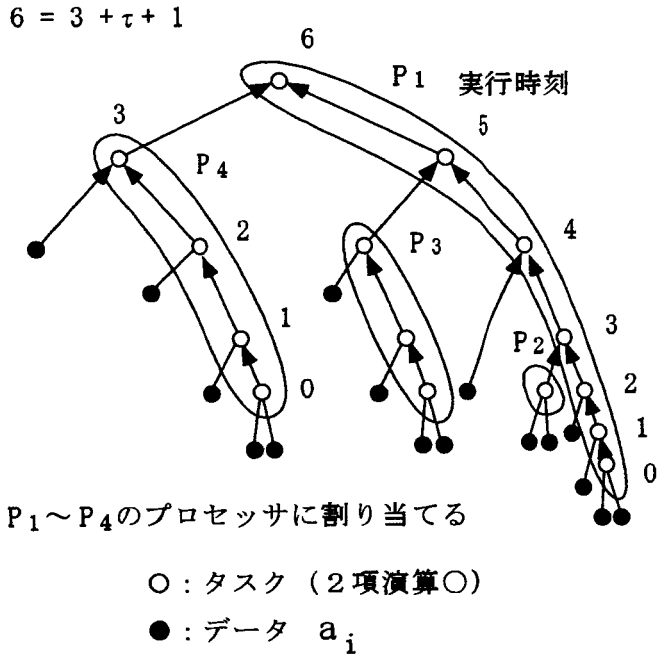


図 4.3: 式 $a_1 \circ a_2 \circ \cdots \circ a_{16}$ を計算する DAG($\tau = 2$)

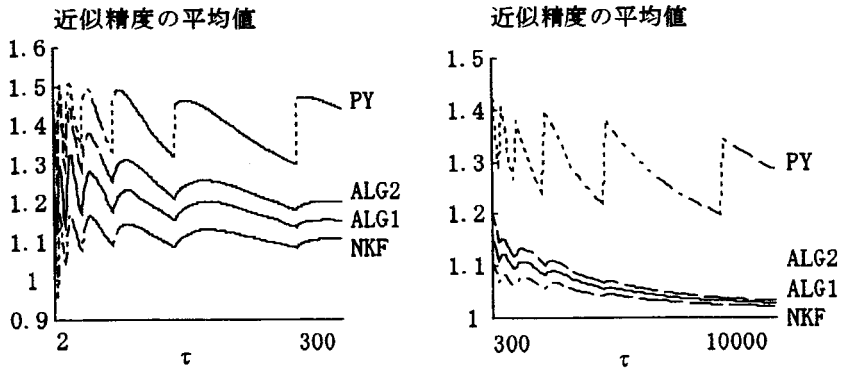


図 4.4: 各アルゴリズムの近似精度の比較（高さ 1 から 20）

当が単純なためにわずかのメイクスパンの増加で必要プロセッサ数を減らせるという利点がある.

最後に利用できるプロセッサ数が限られる場合の ALG2 の拡張であるアルゴリズム ALG3 を示し，その近似精度が最悪高々$\frac{10}{3}$倍であることを示す.

4.2　メイクスパンの下界

4.2.1　既知の下界

スケジュールのメイクスパンの既知の下界について説明する.

通信遅延τと均質な DAG G のタスクvに対して定まる関数$\mathbf{e_{G,\tau}(v)}$を以下のように定義する. ただし$u_1, u_2, \cdots, u_d$はvのすべての子孫で，$e_{G,\tau}(u_1) \geq e_{G,\tau}(u_2) \geq, \cdots \geq e_{G,\tau}(u_d)$とする.

- $d \leq \tau$のとき$e_{G,\tau}(v) = d$
- $d > \tau$のとき$e_{G,\tau}(v) = e_{G,\tau}(u_{\tau+1}) + \tau + 1$

例えば，図 4.2 で$e_{G,2}(15) = 0, e_{G,2}(7) = 2, e_{G,2}(3) = 3, e_{G,2}(1) = 5$である.

$STSP_\tau$において，タスクvを時刻$e_{G,\tau}(v)$に常に実行開始できるとは限らないが，時刻$e_{G,\tau}(v)$より前に実行開始することはできない [17]. したがってvを時刻$c \cdot e_{G,\tau}(v)$に実行開始できるとすれば，それは最適値のc倍以内である.

vの子孫の数がτ以下のときは，vとそのすべての子孫を1つのプロセッサに割り当てることにより，vは時刻$e_{G,\tau}(v)$に実行開始できる. これは最適時間である. ゆえに，ノードの数が$(\tau+1)$個以下のGの$STSP_\tau$では，すべてのタスクを1台のプロセッサに割り当てるスケジュールが最適解である.

4.2.2　既知の下界の改良

この節では，関数$e_{G,\tau}(v)$をもとに，DAG に対するスケジュールのメイクスパンの下界を改良する.

関数$e_{G,\tau}(v)$は，もともとGとτが与えられたときに，vに対して定まる関数である [17]. ところで，ここでvを固定したときのτの変化に対する$e_{G,\tau}(v)$の変化を考える. 例えばC_5の根ノードの場合，τに関する$e_{G,\tau}(v)$の値は表 4.2.2 のようになる. すなわちvを固定したとき

表 4.2: v を固定したときの τ の変化に対する $e_{G,\tau}(v)$ の変化

	τ									
h	1	2	3	4	5	6	7	8	9	10
3	4	3	4	5	6	6	6	6	6	6
4	6	5	6	7	8	7	8	9	10	11
5	8	6	8	10	12	9	10	11	12	13
6	10	8	10	12	14	13	14	15	16	17
7	12	9	12	15	18	14	16	18	20	22

の $e_{G,\tau}(v)$ の値は τ に関して単調増加ではない．ゆえに以下の補題より，$\tau = 2$ と $\tau = 6,7,8$ のときの下界は，それぞれ $\tau = 1$ のときの下界 8 と $\tau = 5$ のときの下界 12 にまで改良できる．

補題 4.2.1 $\tau = x$ のとき $DAG\ G$ のタスク v を時刻 T 以前には実行できないことがわかっているとする．このとき $\tau = X(> x)$ の場合を考えると，v はやはり時刻 T 以前には実行できない．

証明: $\tau = X$ のとき，v を T 以前に実行できるとする．$\tau = X$ のときの G のスケジュール S は $\tau = x$ のときの G スケジュールでもある．ゆえに $\tau = x$ のとき時刻 T 以前に v を実行するスケジュールが存在するので矛盾．**証明終**

このような考察から一般の DAG について以下の定理が成り立つ．

定理 4.2.1 $STSP_\tau$ において，ノード v は時刻 $E_{G,\tau}(v) = \max_{1 \le x \le \tau}(e_{G,x}(v))$ より前には実行開始できない．

完全 2 分木，FFT グラフについてのこの改良の効果を示す．ALG1 の解の精度は，$e_{G,\tau}(v)$ の値からは平均 1.6 倍以内程度であることしか示せないが、$E_{G,\tau}(v)$ の値からは，平均 1.4 倍以内程度であることを示すことができる．FFT グラフは完全 2 分木を重ね合わせたものなので，FFT グラフの改良は同じ高さの完全 2 分木の場合と一致する．ダイヤモンド [16] を対象として PY を適用した場合も多くの点で改良ができる．

関数 $e_{C_h,\tau}(v)$ のグラフを考えると，完全 2 分木の場合は，$\log(\tau + 2)$ が 2 のベキとなる付近の τ に関して，必ず改良できることがわかる．FFT グラフも同様である．

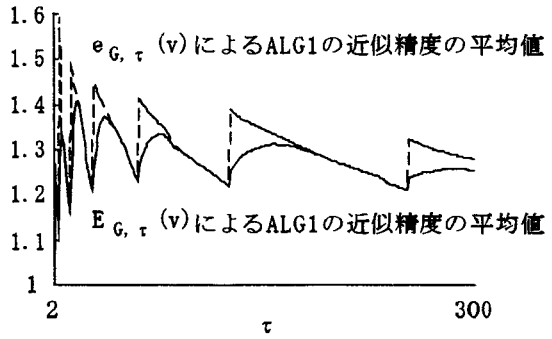


図 4.5: e(v) と改良された下界による ALG1 の精度の比較

4.3　アルゴリズム ALG1

完全2分木の根を r，その子を u，v とする．r と u は同じプロセッサ p で実行し，v は p 以外のプロセッサ q で実行することにすると，p は q から v の結果を通信で受けとらねばならない．このとき p では通信待ちが必要になる場合がある．仮にいま p で通信待ちが必要だったとする．このとき何もせずに v の結果を待つのではなく，(v の子の結果を通信で得てから) v 自身も p が実行した方が，全体として速くなる場合がある．アルゴリズム ALG1 は，このような考え方を一般化したもので，C_h に対するスケジュール S_h を，高さ $h-1$ 以下のスケジュール $S_i(1 \leq i \leq h-1)$ から再帰的に求める．

以下では，あるノード w の i 世代先の子孫を $des_i(w)$ と書く．例えば図 4.2 において，$des_2(3) = 12, 13, 14, 15$ である．また C_h に対する ALG1 の解のメイクスパンを $ALG1_h$ と書く．

アルゴリズム ALG1 は，$h \leq \lfloor \log(\tau+2) \rfloor$ のときは，根の子孫の数が τ 個以下になるので1台のプロセッサにすべてのタスクを割り当てる．$h > \lfloor \log(\tau+2) \rfloor$ のときは，図 4.6 のようにスケジュールする．すなわち，

- 根 r の左部分木 A にスケジュール S_{h-1} を適用する．
- r は A の根 u を実行するプロセッサにおいて実行する．
- 右部分木については，v を根とする高さ j 段の副完全2分木 C に属するタスクをすべて u と同じプロセッサにおいて実行する．残りの副完全2分木 $B_i(1 \leq i \leq 2^j)$ には，スケジュール S_{h-1-j} を独立に適用する．j の値は $0 \leq j \leq h-1$ の範囲で全体のメイクスパンを最小にする値とする．

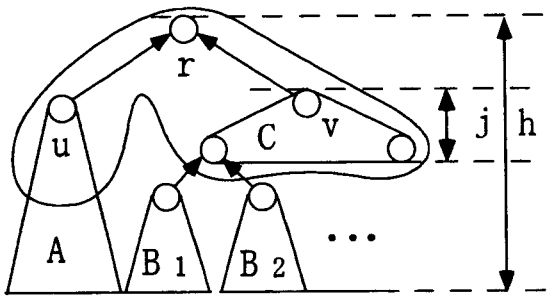


図 4.6: アルゴリズム ALG1 の構成法

u を実行するプロセッサは，u を実行した後，C のタスクの実行に移り，最後に r を実行することになる．このようなスケジュールを生成するアルゴリズムを以下に示す．ただし図 4.2 のように，タスクは根から昇順に $1, 2, 3, \cdots$ と割り当てられた番号で表す．

アルゴリズム 4.3.1 (アルゴリズム ALG1)
入力：均質な C_h と通信遅延 τ
出力：$STSP_\tau$ における C_h のスケジュール
procedure ALG1(h, τ)
  ALG1[1..h]:**array of integer**;
  j[1..h]:**array of integer**;
  p:**integer**;
  procedure genTri(rootTask, h, rootP) **Begin**
    if ($h \leq \lfloor \log(\tau+2) \rfloor$) **then**
      ・rootTask を根とする副木に属するすべての
        タスクを時刻 $ALG1[h] - (2^h - 2)$ から $(2^h - 1)$
        時間の間に rootP 番のプロセッサに割り当てる．
    else begin
      ・rootTask を時刻 $ALG1[h]$ に rootP 番の
        プロセッサに割り当てる．
      ・genTri(rootTask × 2, h-1, rootP);
      ・$startTime = max(ALG1[h-1] + 1,$

$ALG1[h-1-j[h]]+\tau+1)$
・(rootTask × 2+1) を根とする高さ $j[h]$ の
副木 C に属するすべてのタスクを時刻
$startTime$ から $(2^{j[h]}-1)$
時間の間に rootP 番のプロセッサに割り当てる.
・p0 = p; p に副木 B_i の数を加える.
・すべての i に対して
genTri(副木 B_i の根,$h-1-j[h]$,$(p0+i)$);
end
End;
Begin
・p=1 とする.
・$1 \leq i \leq h$ について, ALG1[i] に
数列 $ALG1_i$ の値を代入する.
・$\lfloor \log(\tau+2) \rfloor < i \leq h$ について,
j[i] に高さ h のときの j の値を代入する.
・genTri(1,h,1);
End;

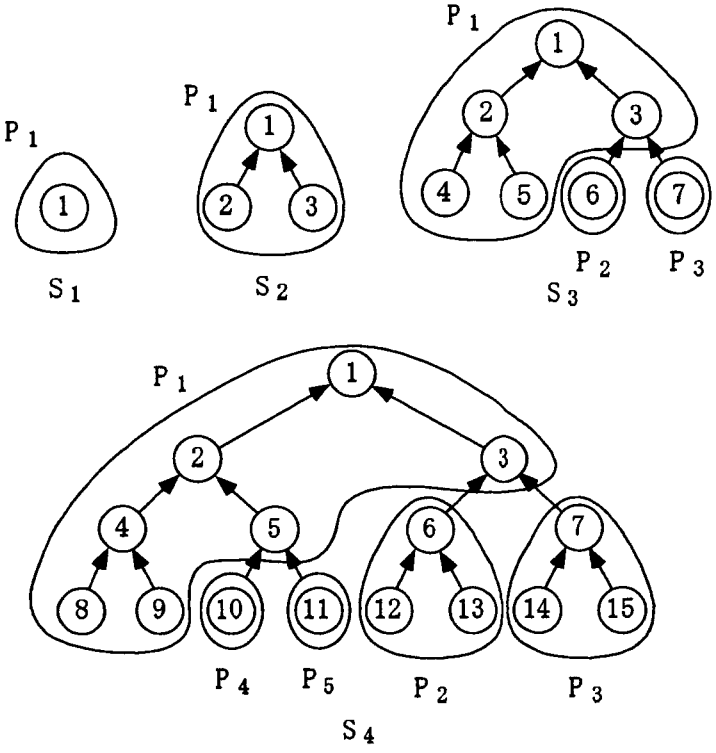
数列 $ALG1_h$ の値は以下のようにして計算できる.

1. 任意の $h(1 \leq h \leq \lfloor \log(\tau+2) \rfloor)$ について $ALG1_h = 2^h - 2$
2. 任意の $h(\lfloor \log(\tau+2) \rfloor < h)$ について
$ALG1_h = \min_{0 \leq j \leq h-1}(\max(ALG1_{h-1}+2^j,$
$ALG1_{h-1-j}+\tau+2^j))$

$\tau = 2$ の場合の例を図 4.7 に示す. 例として, 図 4.7 のスケジュール HF_4 を求めてみる. 手続き P が内部で手続き $Q_1, Q_2, \cdots$ を呼び出すことを $(P,(Q_1,Q_2,\cdots))$ と書くことにする. 図 4.7 のスケジュール HF_4 は, HF(4,2) を実行することにより得られる. 手続き ALG1 の第 1 ステップで HF[1]=0,HF[2]=2,HF[3]=4,HF[4]=6 が得られ, 第 2 ステップで j[3]=1,j[4]=1 が得られるので,

(HF(4,2),(genTri(1,4,1)))
(genTri(1,4,1),(genTri(2,3,1),genTri(6,2,2),genTri(7,2,3)))
(genTri(2,3,1),(genTri(4,2,1),genTri(10,1,4)genTri(11,1,5)))

の手続き呼び出しにより, 図 4.7 のスケジュール HF_4 が得られる.



時刻 / プロセッサ	0	1	2	3	4	5	6
P_1	8	9	4	5	2	3	1
P_2	10						
P_3	11						
P_4	12	13	6				
P_5	14	15	7				

図 4.7: アルゴリズム ALG1 のスケジュールの例（$\tau = 2$）

ALG1の計算時間は$O(n^2)$である．数列$ALG1_h$の値は以下のようになるので$O(\log^2 n)$時間で計算できる．

1. 任意の$h(1 \leq h \leq \lfloor \log(\tau+2) \rfloor)$について$ALG1_h = 2^h - 2$
2. 任意の$h(\lfloor \log(\tau+2) \rfloor < h)$について
   $ALG1_h = \min_{0 \leq j \leq h-1}(\max(ALG1_{h-1} + 2^j,$
   $ALG1_{h-1-j} + \tau + 2^j))$

$ALG1_h$の値を用いて，図4.6をもとに再帰呼び出しを行なう手続きを用いれば，$O(n^2)$時間でALG1を計算できる．

4.4　アルゴリズムALG2

アルゴリズムALG2は図4.8のように均質なC_hを葉から等間隔に分割し，その結果できる副完全2分木の1つ1つに1つのプロセッサを割り当てる方法である．図4.8において，各$h_i(1 \leq i \leq m)$は高々1段しか高さが異ならず，$h_1 \leq h_2 \leq \cdots \leq h_m$である．ここで$m$は$1 \leq m \leq h-1$の範囲で全体のメイクスパンを最小にする値を選ぶ．

アルゴリズムPYは均質なC_hを根から高さ$\lfloor \log(\tau+2) \rfloor$段毎に分割するが，根を含む部分の高さが他と2段以上異なることがある．例えば$\tau = 254$のとき高さ17の完全2分木を葉の方から高さ8，8，1段の3層に分割する．しかし，同じ3層に分割するならば，ALG2のように，6，6，5段と，できるだけ等間隔に分割する方がメイクスパンは小さくなる．このように，アルゴリズムALG2の解のメイクスパンがアルゴリズムPYの解のメイクスパンを越えることはない．

mの値を決定するのに$O(\log n)$時間，mの値に対してスケジュールを生成するのに$O(n)$時間かかるので，ALG2の計算時間は$O(n)$時間である．

4.5　アルゴリズムの比較

4.5.1　スケジュールのメイクスパンの比較

アルゴリズムALG1，ALG2とPYの解のメイクスパンについては図4.4で既に示した．図のPYの近似精度の変化が不連続であるのは，$\log(\tau+2)$が整数でない場合は，PYの直接の適用によるスケジュールより速い，根から高さ$\lfloor \log(\tau+2) \rfloor$段毎に切るスケジュールを用いているからである．$\log(\tau+2)$が整数の場合は，PYにより根から高さ$\log(\tau+2)$段毎に切る

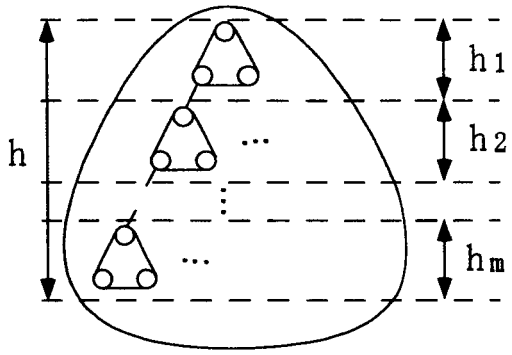


図 4.8: アルゴリズム ALG2 の構成法

スケジュールを得る．$2 \leq \tau \leq 10000$，$\lfloor \log(\tau+2) \rfloor \leq h \leq 20$ の範囲について実験したところ，アルゴリズム ALG1 は常に PY よりもメイクスパンが小さいスケジュールを生成することがわかった．

高さが大きくなるほど ALG1 の解の近似精度は大きくなるが，図 4.4 より，ノード数が 100 万の完全 2 分木でも，300 から 10000 の範囲の整数 τ に対して，平均 1.3 倍以内程度である．

4.5.2 スケジュールの必要プロセッサ数の比較

アルゴリズム ALG1，ALG2 と NKF の生成するスケジュールの必要とするプロセッサ台数を，図 4.4 と同様に調べると図 4.9 のようになる．図には著しく不連続な点が見られるが，これは例えば，$\tau = 8060, h = 13$ のとき，アルゴリズム ALG2 の解はすべてのタスクを 1 台のプロセッサで処理するのに対して，アルゴリズム ALG1 では，図 4.6 で $j = 6$，副木 $A, B_i (1 \leq i \leq 2^j)$ を 1 台のプロセッサで処理することになるので，65 台のプロセッサを用いるからである．この図により，ほとんどの場合，必要プロセッサ数は，ALG2<ALG1<NKF であることがわかる．

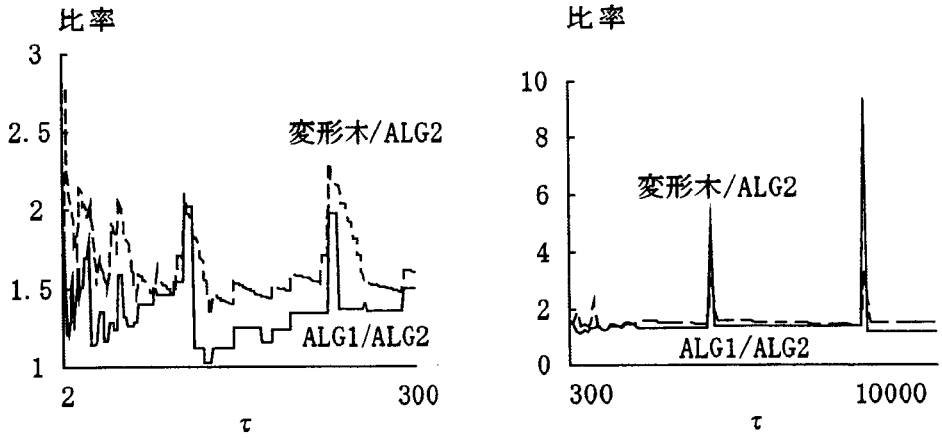


図 4.9: アルゴリズム ALG1, ALG2 と PY の必要プロセッサ数の比率

4.5.3 スケジュールのコストの比較

スケジュールのメイクスパンと必要プロセッサ数の積を，その並列アルゴリズムのコストと定義する．アルゴリズム ALG1, ALG2 と NKF の生成するスケジュールのコストを，図 4.4 と同様に調べると図 4.10 のようになる．著しく不連続な点が見られる理由は，必要プロセッサ数の場合と同じである．この図により，ほとんどの場合，ALG2<ALG1<NKF であるので，コストを考慮すると，ALG2 の解がもっとも優れていることがわかる．

4.6 アルゴリズム ALG3

ALG2 を利用できるプロセッサ数 p が限られる場合へ拡張したアルゴリズム ALG3 を示す．

図 4.11 のように C_h を $(m+1)$ 個の副木に分割することを考える．ALG3 は，p が 2 のべきのときは，C_h を $(p+1)$ 個の副木に分割し，各 $T_i(1 \leq i \leq p)$ に 1 台のプロセッサを割り当て，T_0 は ALG2 で処理する．p が 2 のべきでないときは，$1 \leq h' \leq h_p$ の範囲でメイクスパンが最小になる h' で C_h を分割してスケジュールする．$m \geq p$ のとき $\{T_i | 1 \leq i \leq p\lfloor \frac{m}{p} \rfloor\}$ に p 台のプロセッサをサイクリックに割り当てて，各 T_i を 1 台のプロセッサで処理し，$T_i(p\lfloor \frac{m}{p} \rfloor < i \leq m)$ は $\lfloor \frac{m}{p} \rfloor$ 台のプロセッサで ALG3 を用いて再帰的に処理する．$m < p$ のときは p が 2 のべきのときと同様に処理

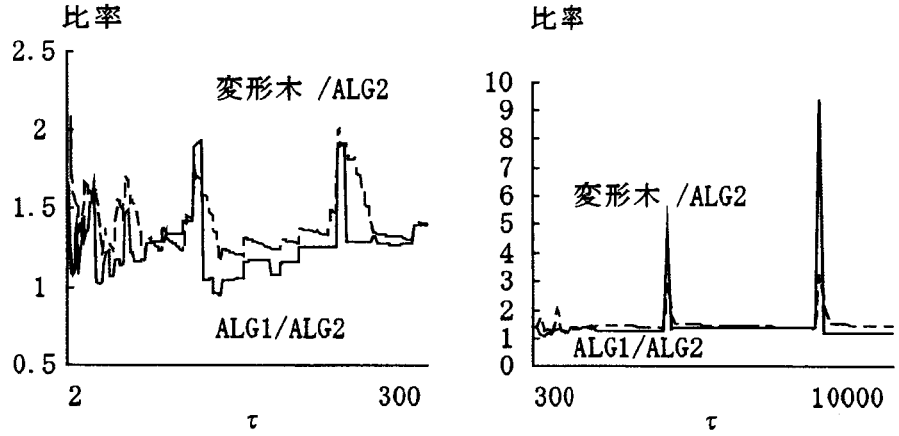


図 4.10: アルゴリズム $ALG1$, $ALG2$ と変形木のコストの比率（高さ 1 から 20）

する．T_0 は ALG2 で処理する．ただし h_p は，p 台のプロセッサで ALG2 で処理できる木の高さの最大値である．プロセッサが足りるときは，全体を ALG2 で処理する．

ALG3 は，並列度が p 以上の部分では，p 台のプロセッサが独立に休むことなく計算を続けるようにし，並列度が p 以下の部分では ALG2 を適用する．p が 2 のべきでない場合は，$p' > p$ である最小の 2 のべき p' に対して C_h を $(p'+1)$ 個に分割するのが基本である．しかし再帰処理の部分で使われないプロセッサがある場合が多いので，最小でない 2 のべき p' も分割の候補としている．また $p' < p$ となる 2 のべき p' で分割すると，スケジュールの並列度は p より落ちるが，T_0 の高さが低くなり，T_0 での通信遅延が減少して，全体として速くなる場合があるので，このような p' も分割の候補としている．

次に ALG3 の解の近似精度を解析的に示す．$h > h_p$ のときの ALG3 の解のメイクスパンを T とすると，$T \leq \frac{n+1}{2p'} \times \lceil \frac{2p'}{p} \rceil + 2e(1+\log p')$．ただし，$p' = \lfloor \log p \rfloor$，$e(h) = e_{C_h,\tau}(r)$ （r は C_h の根）である．$\frac{n+1}{p}$ は下界 $\frac{n}{p}$ の高々 $\frac{4}{3}$ 倍，$2e(1+\log p')$ は下界 $e_{C_h,\tau}(r)$ の高々2 倍であるので，ALG3 の近似精度は高々 $\frac{10}{3}$ 倍である．

$2 \leq \tau \leq 300, 2 \leq p \leq 1024$ の範囲についての ALG3 の近似精度の平均値（$1 \leq h \leq 20$）を解のメイクスパンの下限値から計算した．評価に用いた下限は，$p < \frac{n+1}{2}$ のとき $\max(E_{C_h,\tau}(v), \lceil (n+1)/(2^{1+\lfloor \log p \rfloor}) \rceil + \lfloor \log p \rfloor - 1)$，$p \geq \frac{n+1}{2}$ のとき $\max(E_{C_h,\tau}(v), \log(n+1) - 1$ である．下

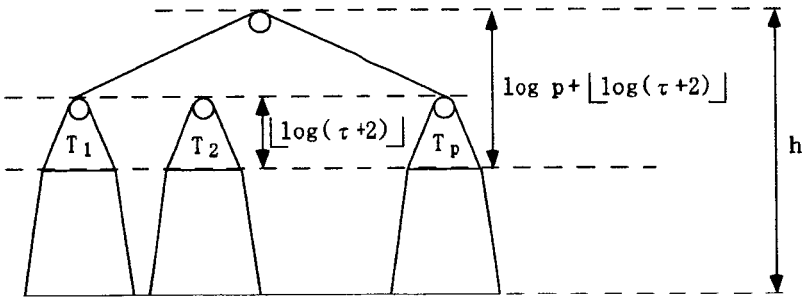


図 4.11: アルゴリズム ALG3 の構成法

限 $(\lceil (n+1)/(2^{1+\lfloor \log p \rfloor}) \rceil + \lfloor \log p \rfloor - 1)$ は文献 [5] の方法を C_h に適用すると得られる．計算により近似精度は平均高々2倍程度，最悪の場合でも高々3倍程度に収まることがわかっている．

ALG2 の計算時間が $O(n)$ だから，ALG3 の計算時間は $O(n \log n)$ である．

4.7 アルゴリズム ALG1 の計算時間の改良

4.3 節での議論から以下の補題が成り立つ．

補題 4.7.1 数列 $ALG1_h$ の値は以下のようにして計算できる．

1. 任意の $h(1 \leq h \leq \lfloor \log(\tau+2) \rfloor)$ について $ALG1_h = 2^h - 2$
2. 任意の $h(\lfloor \log(\tau+2) \rfloor < h)$ について
   $ALG1_h = \min_{0 \leq j \leq h-1}(\max(ALG1_{h-1} + 2^j,$
   $ALG1_{h-1-j} + \tau + 2^j))$

アルゴリズム ALG1 の計算時間を小さくするのに有効な性質として，アルゴリズム ALG1 では，$1 \leq j \leq \lfloor \log(\tau+2) \rfloor + 2$ の場合しか調べる必要がないことを2つの補題により示す．まず最初の補題を証明するための準備を行う．

補題 4.7.2 任意の高さ h の完全2分木の最適スケジュールにおいて，根ノードの実行開始時刻を OPT_h とすると，$\tau \geq 1$ のとき $OPT_{h+1} \geq OPT_h + 2$

証明: 高さ $h+1$ の完全2分木を最適にスケジュールすることを考える．入力完全2分木の根を r，その子を u，v とする．OPT_h の定義より，u,

v は時刻 OPT_h より前には実行を開始できない．また明らかに r は u，v より先には実行を開始できない．ゆえにどのようにスケジュールしようとも時刻 OPT_h には少なくとも r，u，v の３つのタスクが実行できずに残っている．仮定より $\tau \geq 1$ なので，この３つのタスクの実行には３単位時間はかかるから，

$$OPT_{h+1} + 1 \geq OPT_h + 3$$

ゆえに補題成立．**証明終**

定理 4.7.1 $1 \leq \tau \leq 2$ のとき，アルゴリズム ALG1 は最適スケジュールを出力し，$ALG1_h = 2(h-1)$ である．

証明: アルゴリズム ALG1 で $j = 1$ の場合を考える．

- $1 \leq \tau < 2$ のとき $\lfloor \log(\tau+2) \rfloor = 1$ なので，補題 4.7.1 より，$ALG1_1 = 0$
  $h \geq 3$ のとき，補題 4.7.2 より，$\tau \leq 2$ のとき $ALG1_{h-1} + 2^1 \geq ALG1_{h-2} + \tau + 2^1$
  ゆえに補題 4.7.1 より，$h \geq 3$ のとき，$\tau \leq 2 \Rightarrow ALG1_h = ALG1_{h-1} + 2$．ゆえに任意の h に対して $1 \leq \tau \leq 2$ のとき $ALG1_h = 2(h-1)$
  補題 4.7.2 より，これは最適である．

- $\tau = 2$ のとき $\lfloor \log(\tau+2) \rfloor = 2$ なので，補題 4.7.1 より，$ALG1_1 = 0$，$ALG1_2 = 2$．ゆえに 4.7 の場合と同様．

$j = 1$ の場合に最適解が得られるので，他の場合を調べる必要はない．**証明終**

この定理の証明より以下の系を得る．

系 4.7.1 $1 \leq \tau \leq 2$ のとき，アルゴリズム ALG1 で $j = 0$ の場合を調べる必要はない．

これで最初の補題を証明する準備ができた．

補題 4.7.3 $\tau \geq 1$ のとき，アルゴリズム ALG1 で $j = 0$ の場合を調べる必要はない．

証明: $j = 0$ のときと $j = 1$ のときを比較する．補題 4.7.1 より，明らかに $j = 0$ のとき $ALG1_h = ALG1_{h-1} + \tau + 1$ かつ $j = 1$ のとき $ALG1_h = \max(ALG1_{h-1} + 2, ALG1_{h-2} + \tau + 2)$

- $ALG1_{h-1} + 2 \geq ALG1_{h-2} + \tau + 2$ のとき
  $\tau = 1$ のときは，系 4.7.1 より，$j = 0$ の場合を調べる必要はない．
  $\tau > 1$ のとき明らかに $ALG1_{h-1} + 2 < ALG1_{h-1} + \tau + 1$

- $ALG1_{h-1}+2 < ALG1_{h-2}+\tau+2$ のとき
  $ALG1_{h-2}+\tau+2 < ALG1_{h-1}+\tau+1$ すなわち，
  $ALG1_{h-2}+2 \leq ALG1_{h-1}$ であればよい．
  アルゴリズム ALG1 の構成法（図 4.6）を考えると，補題 4.7.2 の証明と同様の理由で，これは常に成立する．

ゆえに $j=1$ のときの方が $j=0$ のときより $ALG1_h$ の候補の値が常に小さいので，補題成立．**証明終**

この補題の証明より以下の系を得る．

系 4.7.2 $\tau > 1$ のとき $ALG1_{h+1} < ALG1_h + \tau + 1$

補題 4.7.4 アルゴリズム ALG1 で $j > \lfloor \log(\tau+2) \rfloor + 2$ の場合を調べる必要はない．

証明: $U = \lfloor \log(\tau+2) \rfloor$ とする．$j=U$ のときと $j>U$ のときを比較する．$\max(ALG1_{h-1}+2^j, ALG1_{h-1-j}+\tau+2^j) \leq \max(ALG1_{h-1}+2^U, ALG1_{h-1-U}+\tau+2^U)$ であるような $j(>U)$ が存在すると仮定する．

1. $ALG1_{h-1-U}+\tau+2^U \geq ALG1_{h-1-j}+\tau+2^j$ のとき
   - $ALG1_{h-1}+2^j \geq ALG1_{h-1-j}+\tau+2^j$ のとき
     $ALG1_{h-1}+2^j > ALG1_{h-1}+2^U$ なので矛盾
   - $ALG1_{h-1}+2^j < ALG1_{h-1-j}+\tau+2^j$ のとき
     $ALG1_{h-1}+2^U < ALG1_{h-1}+2^j < ALG1_{h-1-j}+\tau+2^j$ なので矛盾
2. $ALG1_{h-1-U}+\tau+2^U < ALG1_{h-1-j}+\tau+2^j$ のとき
   - $ALG1_{h-1}+2^j \geq ALG1_{h-1-j}+\tau+2^j$ のとき
     $ALG1_{h-1-U}+\tau+2^U \geq ALG1_{h-1-j}+\tau+2^j$ となる．$\Leftrightarrow$ $ALG1_{h-1-U}-ALG1_{h-1-j} \geq 2^j-2^U$ を得る．ここで $j=U+d(d\geq 1)$ とすると，$ALG1_{h-1-U}-ALG1_{h-1-(U+d)} \geq 2^U(2^d-1)$ となる．ところが系4.7.2より $ALG1_{h-1-U}-ALG1_{h-1-(U+d)} < d(\tau+1)$ である．ゆえに $d \geq 3$ のとき矛盾．
   - $ALG1_{h-1}+2^j < ALG1_{h-1-j}+\tau+2^j$ のとき
     $ALG1_{h-1-U}+\tau+2^U \geq ALG1_{h-1-j}+\tau+2^j$ となるので，上の場合と同様．

ゆえに背理法により $j > \lfloor \log(\tau+2) \rfloor + 2$ のとき $\max(ALG1_{h-1}+2^j, ALG1_{h-1-j}+\tau+2^j) > \max(ALG1_{h-1}+2^U, ALG1_{h-1-U}+\tau+2^U)$
ゆえに補題 4.7.1 より補題成立．**証明終**

補題 4.7.1，4.7.3，4.7.4 より次の定理が得られる．

定理 4.7.2 1. $1 \leq h \leq \lfloor \log(\tau+2) \rfloor$ のとき $ALG1_h = 2^h - 2$

2. $\lfloor \log(\tau+2) \rfloor < h$ のとき
$ALG1_h = \min_{1 \leq j \leq \lfloor \log(\tau+2) \rfloor + 2}(\max(ALG1_{h-1} + 2^j, ALG1_{h-1-j} + \tau + 2^j))$

定理 4.7.3 アルゴリズム ALG1 の計算時間は $O(n + (\lfloor \log(\tau+2) \rfloor) \log n)$

証明: アルゴリズム 4.3.1 を用いて数列 $ALG1_h$ と j_h だけを求めることを考える．アルゴリズム 4.3.1 の再帰はテーブルを用いてループに変換できるので，数列 $ALG1_h, j[h]$ だけなら $O((\lfloor \log(\tau+2) \rfloor) \log n)$ 時間で求まる．ゆえにアルゴリズム 4.3.1 の第 3 ステップまでを $O((\lfloor \log(\tau+2) \rfloor) \log n)$ 時間で実行できる．アルゴリズム 4.3.1 の第 4 ステップが $O(n)$ 時間で実行できることは容易に確かめられる．**証明終**

アルゴリズム ALG1 の生成するスケジュールが必要とするプロセッサ台数は明らかに以下のようになる．

定理 4.7.4 スケジュール $ALG1_h$ の必要とするプロセッサ数を PE_h，高さ h でのアルゴリズム ALG1 の j の値を $j[h]$ とすると，

- $h \leq \lfloor \log(\tau+2) \rfloor$ のとき $PE_h = 1$
- $h > \lfloor \log(\tau+2) \rfloor$ のとき $PE_h = PE_{h-1} + 2^{j[h]} \times PE_{h-1-j[h]}$

4.8 アルゴリズム ALG1 と PY の解析的比較

この章ではアルゴリズム ALG1 の実行時間が PY の実行時間よりも常に短いことを解析的に示す．まずその準備として PY の解の実行時間を調べる．C_h に対する PY の解の実行時間を PY_h と書く．数列 PY_h に関して以下の補題が成立する．

補題 4.8.1 1. h が $\lfloor \log(\tau+2) \rfloor$ の倍数のとき，$PY_{h+1} = PY_h + \tau + 1$

2. h が $\lfloor \log(\tau+2) \rfloor$ の倍数でないとき，$PY_{h+1} = PY_h + 2^i$
ただし $i = (h \ mod \ (\lfloor \log(\tau+2) \rfloor))$.

証明:

- $\log(\tau+2)$ が整数のとき
完全２分木を根から高さ $\log(\tau+2)$ 段毎に切ってできる高さ $\log(\tau+2)$ の完全２分木の１つ１つにプロセッサを割り当てることになる．しかし葉から高さ $\log(\tau+2)$ 段毎に切っても r の実行開始時間は変わらない．ゆえに補題の成立は容易に確かめられる．

- $\log(\tau+2)$ が整数でないとき
  完全2分木の根 r を考える．r の子孫を番号の大きい順に $u_1, u_2, \cdots$ とすると，r と u_τ，$u_{\tau+1}$ の関係は図 4.12 のようになる．アルゴリ

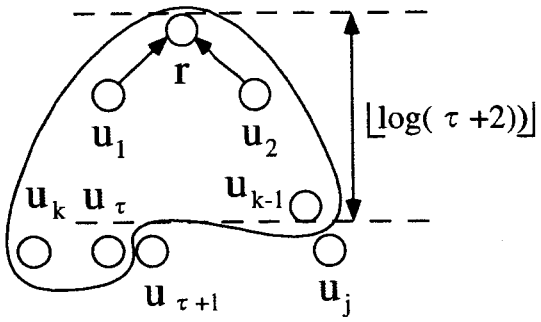


図 4.12: r,$u_\tau u_{\tau+1}$ の関係

ズム PY では，r と $u_1, u_2, \cdots, u_\tau$ を1つのプロセッサに割り当て，$u_{\tau+1}$ の実行開始から $(\tau+1)$ 時間後に実行開始するようにする．しかし $u_k, u_{k+1}, \cdots, u_\tau$ は $u_{\tau+1}$ と同じ高さの完全2分木の根であるので，プロセッサが必要なだけ使えれば，$u_{\tau+1}$ と同じ時刻に実行開始可能である．ゆえに図 4.12 のようにスケジュールするより，r と $u_1, u_2, \cdots, u_{k-1}$ を1つのプロセッサに割り当て，$u_k, u_{k+1}, \cdots, u_\tau, \cdots, u_j$ を根とする完全2分木を $u_{\tau+1}$ を根とする完全2分木と同じように独立にスケジュールして，それらの実行時間の $\tau+1$ 時間後に r を根とする高さ $\lfloor\log(\tau+2)\rfloor$ の完全2分木の実行を開始する方が速くなる．すなわち完全2分木を根から高さ $\lfloor\log(\tau+2)\rfloor$ 段毎に切っていくことになるが，葉から高さ $\lfloor\log(\tau+2)\rfloor$ 段毎に切っても r の実行開始時間は変わらない．
ゆえに先の場合と同様に補題の成立は容易に確かめられる．

証明終

定理 4.8.1 $\tau > 1$ のとき，

1. $1 \le h \le \lfloor\log(\tau+2)\rfloor$ のとき $ALG1_h = PY_h$
2. $\lfloor\log(\tau+2)\rfloor < h$ のとき $ALG1_h < PY_h$

証明: 1 はアルゴリズム ALG1，PY より明らかなので 2 について証明する．証明は帰納法による．

1. $h = \lfloor \log(\tau+1) \rfloor + 2$ のとき
   $1 \le h \le \lfloor \log(\tau+2) \rfloor$ のとき $ALG1_h = 2^h - 2$ だから，$ALG1_{h+1} = \min_{1 \le j \le \lfloor \log(\tau+2) \rfloor + 2}(\max(ALG1_h + 2^j, ALG1_{h-j} + \tau + 2^j))$
   $= \min_{1 \le j \le \lfloor \log(\tau+2) \rfloor + 2}(\max(2^h - 2 + 2^j, 2^{h-j} - 2 + \tau + 2^j))$
   ここで $y = 2^h - 2 + 2^j$ と $y = 2^{h-j} - 2 + \tau + 2^j$ のグラフを考えると図 4.13 のようになる．ゆえに $ALG1_{h+1} < ALG1_h + \tau + 1$．また

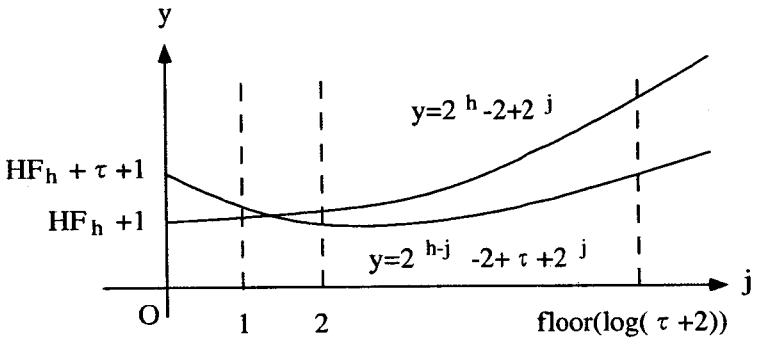


図 4.13: $y = 2^h - 2 + 2^j$ と $y = 2^{h-j} - 2 + \tau + 2^j$ のグラフ

   $PY_{h+1} = PY_h + \tau + 1$ なので $ALG1_h = PY_h$ より $ALG1_h < PY_h$
2. $h = k$ のとき $ALG1_h < PY_h$，$h < k$ のとき $ALG1_h \le PY_h$ とすると，$h = k+1$ のとき
   入力完全2分木の根を r とする．入力完全2分木にアルゴリズム PY を適用したとき，r を根とする高さ $i+1$ の完全2分木に属するノードが1つのプロセッサに割り当てられたとする．$(0 \le i < \lfloor \log(\tau+2) \rfloor - 1)$
   (a) $i = 0$ のとき
   補題 4.8.1 より，$PY_{k+1} = PY_k + \tau + 1$．系 4.7.2 より，$ALG1_{k+1} < ALG1_k + \tau + 1$．ゆえに仮定より $ALG1_k < PY_k$ なので，$ALG1_{k+1} < PY_{k+1}$
   (b) $i > 0$ のとき
   補題 4.7.3，4.7.4 より，アルゴリズム ALG1 での j の値は $1 \le j \le \lfloor \log(\tau+2) \rfloor + 2$ の範囲をとる可能性があるが，ここでは図 4.14 のように，$j = i$ の場合を考える．明らかに $ALG1_{k+1} \le \max(ALG1_k + 2^i, ALG1_{k-i} + \tau + 2^i)$
   i. $ALG1_k + 2^i \ge ALG1_{k-i} + \tau + 2^i$ のとき
   補題 4.8.1 より $PY_{h+1} = PY_h + 2^i$ なので，$PY_{k+1} - ALG1_{k+1} \ge (PY_k + 2^i) - (ALG1_k + 2^i) = PY_k - ALG1_k > 0$

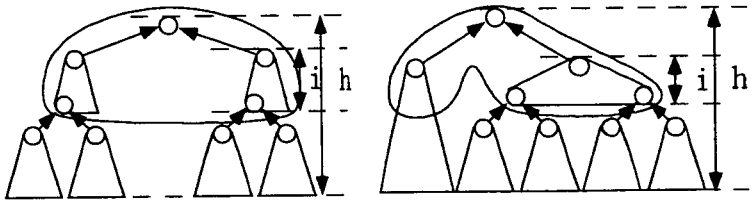


図 4.14: アルゴリズム ALG1

ii. $ALG1_k + 2^i < ALG1_{k-i} + \tau + 2^i$ のとき
図4.14より $PY_{h+1} = PY_{h-i} + \tau + 1 + 2^{i+1} - 2$ なので，$PY_{k+1} - ALG1_{k+1} \geq (PY_{k-i} + \tau + 1 + 2^{i+1} - 2) - (ALG1_{k-i} + \tau + 2^i) = (PY_{k-i} - ALG1_{k-i}) + (2^i - 1) > 0$
ゆえに $ALG1_{k+1} < PY_{k+1}$

ゆえに 1，2 より定理成立．**証明終**

第5章　結論

本研究では，任意のプロセッサ間通信遅延時間を単一のパラメータ τ で抽象化したタスクスケジューリング問題に関する２つの問題を扱った．

分散メモリマシンでは，通信の一括化を行わなければ実行速度の速い並列プログラムは得られない．しかし通信の一括化の行いやすい並列プログラムに対応するスケジュールを生成するタスクスケジューリングに関する研究はこれまでなかった．本研究では，まず，通信の一括化を適用しやすいスケジュールを生成するアルゴリズム BCSH を提案した．BCSH の生成するスケジュール通りに動作する並列プログラムを生成し，分散メモリマシン AP1000 上で評価したところ，BCSH を用いて生成した並列プログラムでは，良好な速度向上率が得られることがわかった．

任意のプロセッサ間通信遅延時間を単一のパラメータ τ で抽象化した並列計算機モデルに対して，単位時間のタスクのみからなる完全２分木状のタスクグラフを対象としたタスクスケジューリングアルゴリズム ALG1 と ALG2 を示した．ALG1，ALG2 により既存のアルゴリズムよりも速く，最適解に近い解が得られる．n ノードの完全2分木のスケジュールに必要なプロセッサ数は高々n 台である．さらにスケジュールのメイクスパンの既知の下界を改善して，$2 \leq \tau \leq 10000$ の範囲の通信遅延 τ と高さ h が $1 \leq h \leq 20$ の範囲の完全２分木に対して，アルゴリズム ALG1，ALG2 と PY[17] の近似精度をより正確に評価した．この結果，完全２分木を対象とする場合は，非常に単純な方法によっても最適解に近い実行時間のスケジュールが得られることを示した．さらに利用可能なプロセッサ数が限られる場合に対して ALG2 を拡張したアルゴリズム ALG3 を示した．ALG3 の近似精度は高々$\frac{10}{3}$ 倍であることを示した．

今後の課題としては以下のように考えている．

- アルゴリズム BCSH の独立グループ集合のバランス判定条件と節点グループのマージ条件の改良．
- アルゴリズム ALG3 を応用したリダクション演算ライブラリの作成と評価．

謝辞

本研究の全課程を通じて懇切なる御助言と御指導を賜りました大阪大学大学院基礎工学研究科の萩原 兼一教授に深く感謝の意を表します.

本研究の内容に関し，適切なる御意見を戴きました谷口 健一教授，柏原 敏伸教授に心より感謝致します.

筆者が大阪大学基礎工学部情報工学科および同大学院に在籍中，講義などを通じ様々な御指導と御教示を賜りました，大阪大学大学院基礎工学研究科の井上 克郎教授，今井 正治教授，菊野 亨教授，都倉 信樹教授，故 西川 清史教授，橋本 昭洋教授，東野 輝夫教授，故 藤井 護教授，藤原 融教授，宮原 秀夫教授，村田 正幸教授，同大学産業科学研究所の北橋 忠広教授，同大学医学系研究科機能画像診断学研究部の田村 進一教授，現 大阪工業大学情報科学部 首藤 勝教授に深謝致します.

研究の上で様々な御支援を戴きました滝口美也子事務官をはじめ，萩原研究室の皆様に感謝致します.

関連図書

[1] Ahmad, I. and Kwok, Y. : "On Exploiting Task Duplication in Parallel Program Scheduling", IEEE Transactions on Parallel and Distributed Systems, Vol.9, No.9, pp.872–892, September 1998

[2] Almasi, G. S. and Gottlieb, A. : "Highly Parallel Computing", The Benjamin / Cummings Publishing, 1994

[3] Bacon, D.F., Graham, S.L., and Sharp, O.J.: "Compiler transformations for high-performance computing", acm computing surveys, Vol.26, No.4, pp.345–420, December 1994.

[4] El-Rewini, H., Lewis, T.G., and Ali, H.H.: "TASK SCHEDULING in PARALLEL and DISTRIBUTED SYSTEMS", pp.56–105, PTR Prentice Hall, 1994.

[5] Fernández, E. B. and Bussel, B. : "Bounds on the Number of Processors and Time for Multiprocessor Optimal Schedules",IEEE Trans. on Comput. VOL.C-22, NO.8, pp.745-751 (August 1973)

[6] Gabow, H. : "An almost linear algorithm for two-processor scheduling", J. ACM, Vol.29, No.3, pp.766-780, 1982

[7] Graham, R. L.: "Bounds on multiprocessing timing anomalies", SIAM Journal of Applied Mathematics, Vol.17, pp.416–429, 1969

[8] 本多弘樹, 水野聡, 笠原博徳, 成田誠之助 : "OSCAR 上での Fortran プログラム基本ブロックの並列処理手法", 電子情報通信学会論文誌 D-I, Vol. J73-D-I, No.9, pp.756–766, 1990

[9] Ishihata, H., Horie, T., Inano, S., Shimizu, T. and Kato, S.: "An architecture of highly parallel computer AP1000", Proc. IEEE Pacific Rim Conf. Commun, Comput. Signal process, pp.13–16, 1991

[10] 笠原博徳 : "マルチプロセッサシステム上での近細粒度並列処理", 情報処理学会誌, Vol.37, No.7, pp.651–661, 1990

[11] Kruatrachue, B.: "Static task scheduling and packing in parallel processing systems", Ph.D. diss., Department of Electrical and Computer Engineering, Oregon State University, Corvallis, 1987.

[12] Message Passing Interface Forum: "MPI: A Message-Passing Interface Standard Version 1.1", 1995.

[13] 野々村 洋, 栗野俊一, 深澤良彰 : " 通信遅延を仮定した時の最適なリダクション計算", 情報処理学会第 47 回全国大会, 6E-8 (October 1993)

[14] Ogawa, H. and Matsuoka, S.: "OMPI: optimizing MPI programs using partial evaluation", Proceedings of the 1996 IEEE/ACM Supercomputing Conference, Pittsburgh, November 1996.

[15] Palis, M. A., Liou, J., and Wei, D. S. L. : "Task Clustering and Scheduling for Distributed Memory Parallel Architectures", IEEE Transactions on Parallel and Distributed Systems, Vol.7, No.1, January 1996

[16] Papadimitriou, C. H. and Ullman, J. D. : "A Communication-Time Tradeoff", SIAM J. Comput. vol.16, No. 4, pp.639-646 (August 1987)

[17] Papadimitriou, C. H. and Yannakakis, M. :"Towards An Architecture-Independent Analysis of Parallel Algorithms ", SIAM J. Comput. vol. 19, No. 2, pp.322-328 (April 1990)

[18] Thurimella, R. and Yesha, Y. :"A Schduling Principle for Precedence Graphs with Communication Delay", Internatinal Conference on Parallel Processing III, pp.229-236 (1992)

[19] Skillicorn, D.B., Hill, J.M.D., and McColl, W. F. : "Questions and answers about BSP" , Scientific Programming, 6(3): pp.249–274, 1997.

[20] Valiant, L.G.: "A bridging model for parallel computation", Communications of the ACM, Vol.33, No.8, 1990.

[21] 湯淺太一, 安村通晃, 中田登志之 : "はじめての並列プログラミング", 共立出版, 1998